Ingeniería de Software I

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Práctica: "Diseño Orientado a Objetos"

Departamento de Computación
Facultad de Ciencias Exactas
Universidad de Buenos Aires

Parte I:

Introducción. Nociones básicas de DOO. Realidad. Dominios. Entes. Objetos. Mensajes. Métodos. Comportamiento. Responsabilidades. Clases. Notación y Sintaxis de Diagramas. Diagramas. Jerarquías. Herencia. Encapsulamiento. Polimorfismo. Tipado. Organización de Conocimiento: Clasificación vs Prototipado.

Ejercicio 1 – Nociones Básicas. Dominios. Entes. Objetos.

En los casinos, el termino *juego de mesa* se utiliza para distinguir a juegos que se juegan sobre una mesa y son operados por uno o más croupiers, de aquellos que se juegan en dispositivos mecánicos o electrónicos como las máquinas tragamonedas. El blackjack, el poker, los dados (craps) y la ruleta son ejemplos de juegos de mesa.

Un *juego de cartas* es un tipo de juego donde se utiliza a las cartas como protagonistas principales. Incontables juegos de cartas existen, incluyendo familias de juegos de cartas relacionadas (por ej el poker, con todas sus variaciones).

En el poker se utiliza un mazo de 52 cartas y se reparten 5 a cada jugador al comienzo del juego. Existen 4 palos diferentes (trébol, diamante, pica y corazón) y dentro de cada uno las cartas están rankeadas desde el A (la más alta),K ,Q, J, 10,…, hasta el 2 (el valor más bajo).

El croupier reparte las cartas de a una a cada jugador y de forma intercalada. Cada jugador las recibe y debe utilizar un algoritmo de *Insertion Sort* para mantener su mano ordenada a todo momento.

En base al texto anterior, se pide:

i) Identifique al menos 3 dominios de problemas diferentes.

ii) Nombre ejemplos de modelos computables que apliquen a cada uno de los dominios planteados en i.

iii) En base a lo visto en las clases ¿Qué son una carta, un dado o una ruleta?

iv) Elija un ente cualquiera, y construya una representación del mismo para alguno de los dominios identificados del punto i. ¿Cómo se le llama a esa representación?

v) ¿Qué sería el algoritmo de ordenación Insertion Sort? ¿Dónde vive? ¿Y qué es su código ya escrito en un lenguaje de programación y en ejecución?

Ejercicio 2 – Esencia de los Objetos.

i) Una reconocida cadena de cines desea ofrecer a sus clientes la posibilidad de comprar entradas a las películas que exhiben sus salas de manera online. El cliente registrado debe de tener un "carrito" en el que pueda agregar entradas para cualquiera de sus salas de alguna de sus sucursales en un horario/día particular. Al hacerlo debe de poder elegir sus asientos de cualquiera de los disponibles de esa sala en ese horario y día y seleccionar promociones de una lista de aplicables para obtener descuentos en sus montos. Se debe de contar con la posibilidad de listar la cantidad de entradas de cada película actualmente en el carrito.

Cuando el cliente lo decide pasa a hacer un checkout del carrito, y se deben listar las entradas para cada película con su cantidad, monto a pagar individual y suma total, detallando también los descuentos por promociones elegidas. Cuando se

realiza la venta, se utilizan los datos de la tarjeta de crédito del cliente y se emite un ticket de venta.

Los montos de las entradas pueden variar entre sala y sala, y también por la fecha de la función elegida. Adicionalmente se quiere tener un registro de todas las operaciones realizadas para fines estadísticos.

a) Identifique objetos participantes del problema.

b) Si identificó alguno del estilo "CarritoClienteDNI12345678" detalle sus responsabilidades.

c) ¿Se imagina otros contextos donde el objeto anterior tenga responsabilidades diferentes?

d) Si identificó algún objeto de nombre parecido a "Sala8SucursalCaballito" detalle sus responsabilidades.

e) Detalle responsabilidades de al menos otros dos objetos identificados.

f) Considere un mensaje cualquiera de los que identificó como responsabilidades de alguno de los objetos anteriores. ¿Qué es ese mensaje? Si su respuesta fue "es un objeto", nombre al menos tres mensajes que este sepa responder.

ii) Identifique (de ser posible) los objetos que saben responder a los siguientes mensajes:

a) void play(track)

void stop()

void pause()

track next()

track previous()

b) bool isEmpty()

push(anObject)

anObject pop()

c) bool isEmpty()

add(anObject)

anObject next()

d) void bailar()

e) void jugarGolf(circuito)

conexion conectar(dirección, opciones)

superficie renderizar (modelo3d)

void debitar(monto)

iii) Luego de efectuar ii) revise las responsabilidades asignadas a los objetos identificados en i).

iv) Para los siguientes entes del mundo real:

1. Persona.
2. iPad.
3. Alumno en un sistema de inscripciones.
4. Factura de venta de un libro.
5. Color Verde.
6. Instante de tiempo actual.

7. Alumno en un sistema de asignación de aulas.
8. Puerta de Roble.
9. El Obelisco en Google Maps.
10. Votante en el padrón electoral.

a) Proponga un conjunto de características esenciales que lo identifican unívocamente.
b) ¿Qué ocurre si 1) 2), 7) y 10) representan a la misma persona? Piense en la relación entre un ente y los dominios en los que participa.
c) Piense en cada caso qué efectos traería definir un método que *mute* sus atributos esenciales.
d) Ahora piense en sus implementaciones en una base de datos. ¿Los atributos esenciales son los que definiría como clave primaria en cada caso?
e) Explique con sus palabras porque es importante modelar un ente teniendo en cuenta el rol que ocupa en el dominio de su problema.

Ejercicio 3 – Mensajes Vs. Métodos.

1. ¿Cuál es la diferencia entre mensaje y método?
2. ¿Cuál es la responsabilidad principal de un mensaje? ¿y la de un método?
3. ¿Cuál es la diferencia entre enviar un mensaje e invocar una función en el paradigma estructurado?
4. ¿Qué relación tiene la separación mensaje/método con dynamic binding?
5. ¿Qué relación tiene esta separación con el *polimorfismo*?
6. +-*/ ¿Son mensajes, métodos o funciones? ¿Está pensando en un lenguaje en particular o en el paradigma en sí?

Ejercicio 4 – Mensajes entre objetos.

Para cada uno de los siguientes fragmentos de código Smalltalk:

- Explique detalladamente la interacción entre los objetos participantes (¿Qué objeto envía qué mensaje a qué otro objeto?)
- Escriba un código similar en C++ o C# o Java. ¿Qué necesita utilizar?

i)
```
x:= x>0 ifTrue:[ x+1 ] ifFalse:[ 0 ].
```

ii)
```
1 to: 20 by: 2 do: [:x | Transcript show: x ; space]
```

iii)
```
i := Interval from: 5 to: 10 by: 2.
i do: [:x| Transcript show: x ; space]
```

iv)
```
|i|
i:=5.
[i >0] whileTrue:[
        Transcript show: ((i*2) asString) ; cr.
        i:=i-1.
        ]
```

v)
```
withChecksOver: amount do: aBlock
        history keysAndValuesDo:
        [:key :value |
                (value > amount) ifTrue:
```

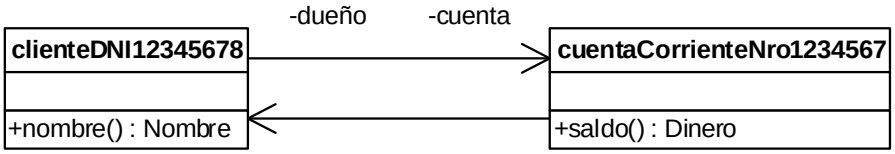
[aBlock value: key]

]
(*history es un diccionario de cheques, con clave el numero del cheque y value el valor del mismo).

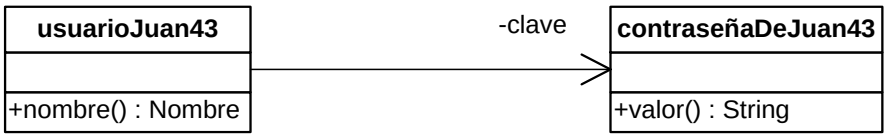
Ejercicio 5 – Mapeo Pseudocódigo – Diagramas I.

Escribir un pseudocódigo (o un código en algún lenguaje de programación que ud. elija) para cada inciso, de un modelo computable que le permita generar los siguientes **diagramas de objetos**: (**Ignore los signos "+" y "-" de los diagramas).**

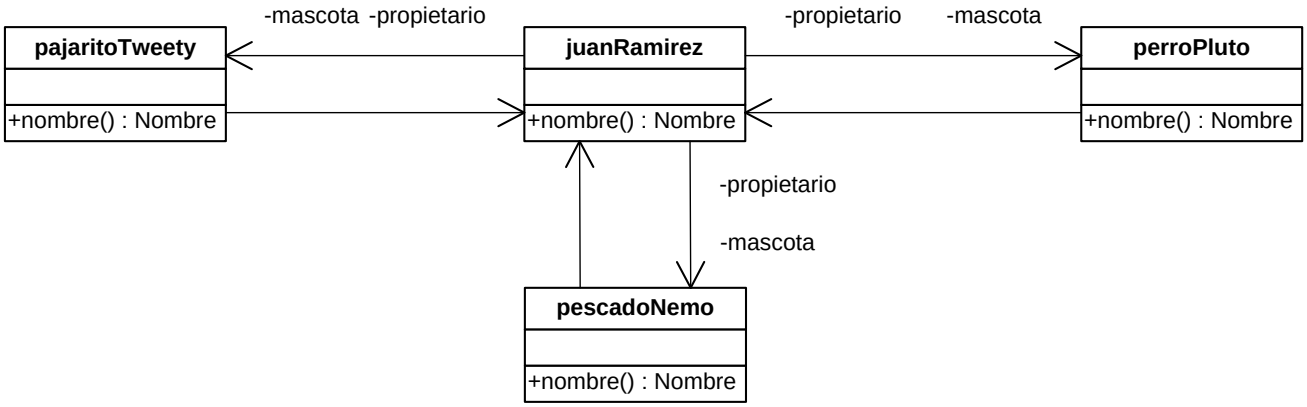
i)



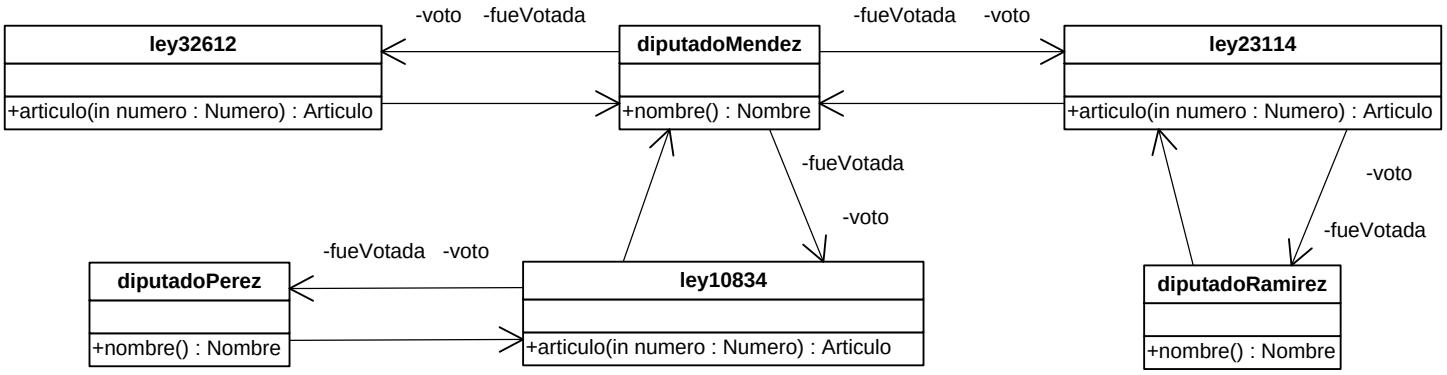
ii)



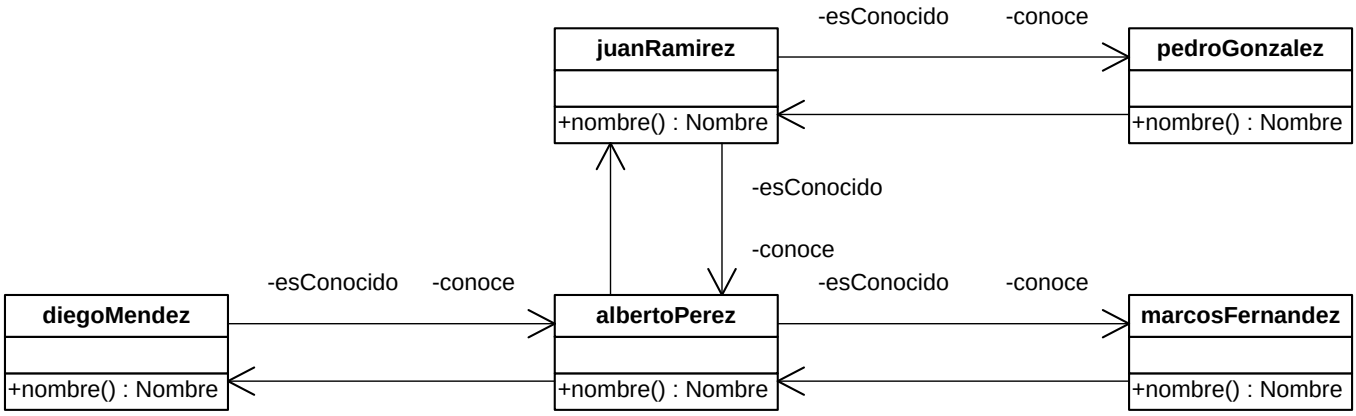
iii)



iv)



v)



Ejercicio 6 – Realidad Subjetiva I.

Modele ***un semáforo*** utilizado para ordenar el tránsito de una esquina típica de Buenos Aires. Por ej. Callao y Santa Fe. Utilice **diagramas de Objetos y de Secuencia** para realizar su tarea.

Hint: Puede seguir los siguientes pasos:

1. ¿Qué entiende ud. por **un** semáforo? ¿Qué entiende ud. por **varios** semáforos?
2. ¿Cuántas luces tiene un semáforo? ¿Cuántas cajas? ¿Cuántos postes?
3. Identifique todos los objetos presentes en el contexto de este problema.
4. Utilice un diagrama de secuencia para mostrar el funcionamiento normal de un semáforo. Comience la secuencia cuando el mismo se prende. Suponga que al prenderse un semáforo realiza un aviso de advertencia que se está encendiendo que consiste en titilar las luces amarillas por 30 segundos.
5. Revise el protocolo de cada objeto: Pensando el nombre con los que los identificó, ¿parecen coherentes los mensajes que puede recibir cada objeto?
6. Si tiene un objeto llamado Semáforo ó LuzVerde vuelva a 1 (o consulte a su ayudante amigo más cercano).
7. Si no tiene un objeto Timer… ¿cómo hizo para que, cada cierto tiempo (ej: 40seg.), cambien las luces del semáforo? Si no tiene ningún objeto que se encargue de esta tarea vuelva a 1.
8. ¿Identifica patrones de colaboración repetidos? ¿Qué le está faltando identificar?

Ejercicio 7 – Realidad Subjetiva II.

Teniendo en cuenta lo ejercitado en el ejercicio 6, ahora modele el sistema que controla un ascensor automático de un edificio de 4 pisos. El mismo atiende los pedidos en orden de llegada, a menos que le quede de paso atender un pedido realizado que no altere el recorrido que tenía previsto. Si cree que le sirve, comience el modelado considerando uno no automático. Más aún, si cree que le sirve, comience el modelado por uno que funciona manualmente con una persona que lo controla desde dentro del mismo. Utilice **diagramas de Objetos y de Secuencia** para realizar su tarea.

Ejercicio 8 – Modelado. Dominio Conocido (?)

Modelar el sistema de inscripción de alumnos de esta carrera.

Tenga en cuenta que es necesario saber que materias aprobó un alumno a una fecha particular, que materias y puntos le falta para terminar la carrera, cual es el promedio de cursada que posee, cuando curso un materia particular y cuáles fueron los finales que dio de una materia dada (con su nota correspondiente). Utilice **diagramas de Objetos y de Secuencia** para realizar su tarea.

Ejercicio 9 - Tipos de Jerarquías.

i) Dadas las siguientes clases: *DispositivoEléctrico*, *RadioAM-FM*, *SonyDigitalRadio*, *Lavarropas*, *Dispositivo*, *DispositivoMecánico* y utilizando el conocimiento intuitivo del dominio del problema, se solicita lo siguiente:

a- Determine el comportamiento esencial que los objetos, instancia de las clases dadas, tienen. **(¿Qué debe hacer?)**

b- Establecer una jerarquía de las clases.

c- ¿En qué se basó para armar su jerarquía?

d- Agregue más clases a la jerarquía.

ii) Ahora, y a partir de las siguientes clases: *Ala, Turbina, Cola, Hélice, Fuselaje, Rueda, Neumáticos, Avión, Hangar* se pide:

a- Establecer una jerarquía de las clases.

b- ¿En qué se basó para armar su jerarquía?

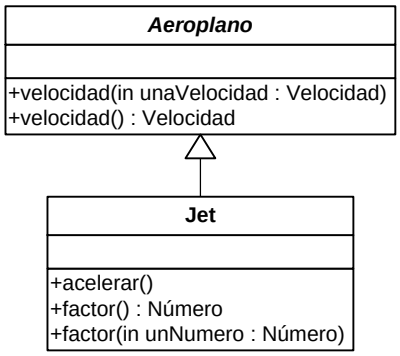
c- Agregue más clases a la jerarquía.

iii) Analice sus resultados de los puntos i) y ii). ¿Qué diferencia encuentra entre las jerarquías encontradas? Escriba un diagrama de clases para cada punto.

Ejercicio 10 – Mapeo Pseudocódigo-Diagramas II.

A partir de una primera versión del modelado de un aeroplano en el siguiente diagrama de clases…

(Ignore los signos "+" de los diagramas)

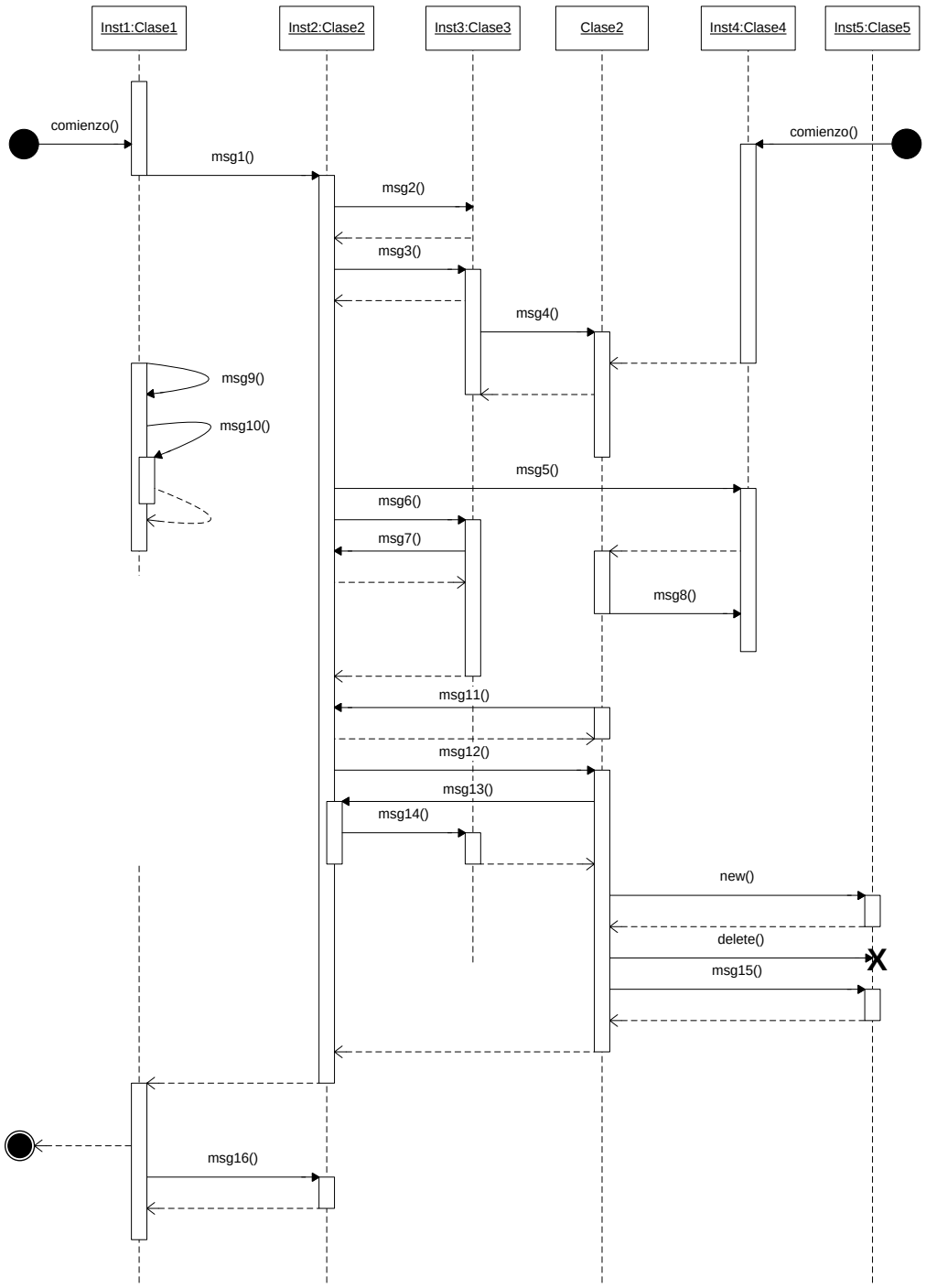


i) Escriba el pseudocódigo del mismo. No ignore ningún aspecto de lo descripto en el diagrama. Para el método *acelerar*() de *Jet* suponga que cada vez que es ejecutado, el mismo incrementa su velocidad en actual *factor* veces.

ii) Complete el diagrama de clases provisto y escriba el pseudocódigo de una clase *PruebaDeVuelos* que crea dos Jets, les coloca velocidades iniciales diferentes, y luego los hace acelerar.

Ejercicio 11 – Sintaxis permitida de diagrama de secuencias.

Dado el siguiente diagrama de secuencias, liste **detalladamente** todos los errores que encuentra.



Ejercicio 12 – Polimorfismo.

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck." -- James Whitcomb Riley

1. ¿Qué es polimorfismo y para qué sirve? ¿El concepto de polimorfismo se aplica a las clases o a los objetos?
2. Dé un ejemplo de polimorfismo en un lenguaje estáticamente tipado (C++, Java).
3. Dé un ejemplo de polimorfismo en un lenguaje dinámicamente tipado (Smalltalk, Python).
4. Dé un ejemplo de polimorfismo en un lenguaje prototipado (self, javascript).
5. ¿Qué ventajas y desventajas encuentra en cada uno?
6. Analice el siguiente código Python:

```
class Pato:
  def cuack(self):
     print("Cuaaaaaack!")
  def plumas(self):
     print("El pato muestra sus plumas blancas y grises.")
  def planchar(self):
     print("El pato flota en la laguna.")

class Persona:
  def __init__(self, nombre):
     self.n = nombre
  def cuack(self):
     print("La persona grita: "Cuaaaaaack!")
  def plumas(self):
     print("La persona toma sus plumas de entre sus lapiceras y
     lápices y las muestra.")
  def planchar(self):
     print("La persona se pone a planchar su ropa recién lavada.")
  def nombre(self):
     print self.n

def enElBosque(pato):
  pato.cuack()
  pato.plumas()
  pato.planchar()

def juego():
  duffy = Pato()
  juan = Persona("Juan Perez")
  enElBosque(duffy)
  enElBosque(juan)

juego()
```

   Luego, escriba exactamente el mismo comportamiento pero utilizando un lenguaje estáticamente tipado como Java. ¿Qué diferencias, ventajas y desventajas encuentra en cada versión?

Ejercicio 13 – Herencia. Noción de clase abstracta.

i) En una colmena viven abejas, hay obreras, una reina y zánganos. Las abejas comen determinadas cantidades de miel, polen o jalea real. Las obreras alimentan a las larvas. Todas las abejas vuelan, en particular, cuando la abeja reina esté

volando podrá ser fecundada por aquel zángano que logre alcanzarla y los demás caerán.

a- Reconocer objetos y responsabilidades.

b- Reconocer clases y construir jerarquías de herencia.

c- Observamos que la abeja reina vuela desde la base de la colmena hacia la cima de esta en círculos concéntricos hasta ser alcanzada por un zángano, las obreras vuelan en forma helicoidal (como un resorte) mientras que los zánganos vuelan de a trayectos rectos.

Utilice todas las técnicas que considere necesarias.

No realizar suposiciones sobre el dominio. Limitarse a los escenarios descriptos.

ii) Proseguimos explorando el dominio del problema y nos damos cuenta que existen obreras recargadas que además de alimentar larvas (tal como una no recargada) salen de la colmena a recolectar alimento.

a- ¿Cómo afecta esto la jerarquía de clases propuesta en i)?

b- Descubrimos que las obreras recargadas vuelan en forma senoidal a diferencia de las no recargadas que lo hacen en forma helicoidal.

c- Nos enteramos que, en la jerga de los apicultores, a las obreras no recargadas se las llama holgazanas.

Ejercicio 14 – Uso de interfaces. Herencia múltiple (problema del diamante).

Realice un diagrama de clases/objetos para cada uno de los siguientes ítems:

- **Suponga en cada caso primero que está diseñando en un ambiente que posee chequeo de tipos dinámico, y luego en uno de chequeo de tipos estático. ¿Hay diferencias?**

i) Un objeto "pintor" sabe colorear de un color cosas que pueden tener un color. Hay figuras geométricas como el rectángulo y el circulo que pueden colorearse, pero otras como el triangulo que no pueden.

ii) Dieguito es un chico al que le gustan muchos los deportes. De hecho toma clases de Fútbol, Tenis y Natación con diferentes entrenadores. Un futbolista sabe patear una pelota, un tenista sabe golpearlas con una raqueta y un nadador sabe nadar dada una pileta. Un entrenador de fútbol sólo sabe entrenar a futbolistas pero no sabe nada (ni le interesa) de otros deportes, ni del resto de la vida de los chicos fuera de su vida como futbolistas. Lo mismo ocurre con los entrenadores de los otros deportes.

iii) Al participar de un concurso de canto, un participante (cantante) debe en algún momento cantar. Al participar de un concurso de baile, un participante (bailarín) debe en algún momento bailar. En los concursos, los participantes pueden no necesariamente ser personas. Por ej. si bien una persona puede cantar y bailar, un canario sólo puede cantar y un oso sólo puede bailar.

Ejercicio 15 - Organización del Conocimiento. Clasificación vs Prototipado.

Resuelva los siguientes problemas utilizando primero clasificación y luego prototipado. Utilice diagramas de objetos/clases y escriba el pseudocódigo de todos los métodos involucrados.

i) Factorial.

ii) Fibonacci.

iii) Algebra booleana (Operadores lógicos AND, OR, y NOT).

Parte II:

Principios de Diseño. Inmutabilidad. Acoplamiento y Cohesión. Dependencias. Noción de "buen" modelado = "buen" diseño. Análisis Crítico.

Ejercicio 1 – Objetos inmutables.

A partir de la siguiente descripción:

En una tienda de electrodomésticos, cada vez que se concreta una venta de productos, se confecciona una factura. En cada línea de la misma se detalla la descripción del artículo vendido, la cantidad vendida del mismo y su importe individual en pesos. La factura además posee el cálculo del total del importe a abonar por el cliente.

Se pide:

i) Identifique las responsabilidades de los objetos del texto anterior y efectué un diagrama de objetos/clases y el pseudocódigo de los mismos.

Hint: Preste especial atención a los mensajes de creación de instancia, setters y getters.

ii) ¿Y si se desea cancelar una factura?

Ejercicio 2 - Análisis Crítico.

Había una vez un feliz programador Java que trabajaba en una empresa implementando un sistema bancario. El día había comenzado como cualquier otro pero lo que él no sabía es que estaba a punto de encontrarse con su peor pesadilla…

Ese día comenzaría la tarea de implementar la cuentas bancarias de los clientes del banco. Luego de leer la documentación correspondiente, decide empezar a "tirar código".

"Una de las responsabilidades de una cuenta bancaria es responder su fecha de creación" - piensa - "por lo tanto parece razonable que la clase CuentaBancaria tenga un mensaje fechaDeCreación que será representado por un objeto de la clase … ¿de qué clase?"

Una rápida búsqueda en Google "java date" lo lleva a:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Date.html

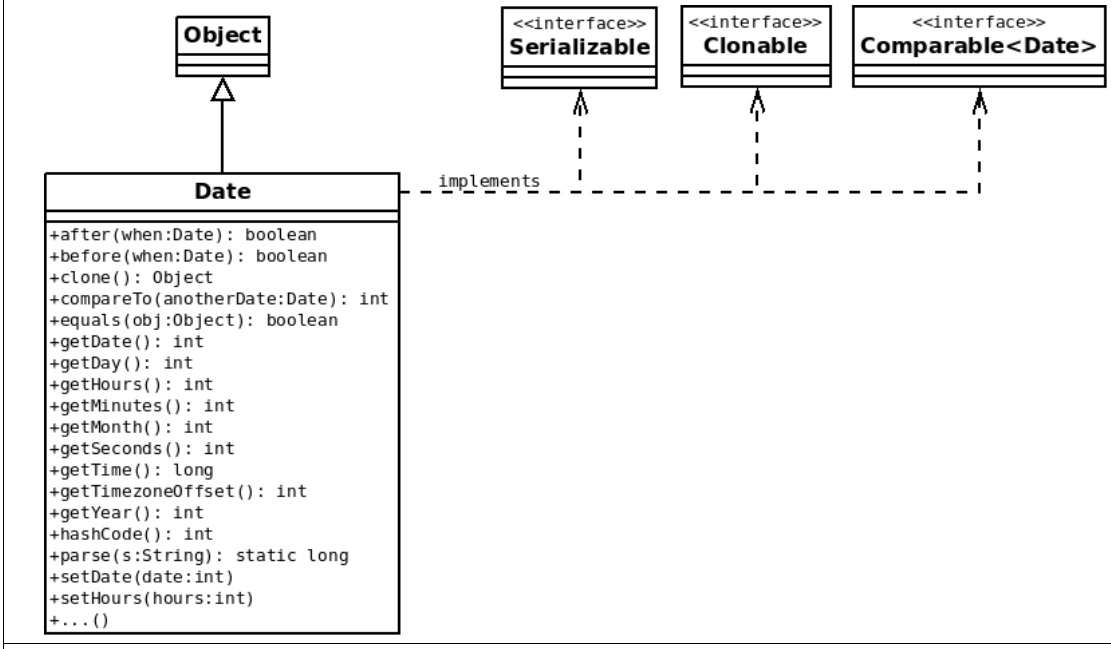
Para su sorpresa una primera mirada por la documentación lo deja confundido. Se detiene y relee el siguiente párrafo:

> In all methods of class Date that accept or return year, month, date, hours, minutes, and seconds values, t following representations are used:
>
> - A year *y* is represented by the integer *y* - 1900.
> - A month is represented by an integer from 0 to 11; 0 is January, 1 is February, and so forth; thus 11 is December.
>
> …

1. ¿Qué opina de estas notas?
2. ¿Es correcto que un modelo de representación de fechas explique que un año o un mes se representa con un **integer**? ¿Qué estaba esperando como representación?
3. ¿Qué espera que ocurra en caso que se intente construir una fecha con una representación inválida? ¿Dónde va a ocurrir el error? ¿Cómo espera encontrarlo?
4. ¿A qué le hace reflexionar los comentarios "the usual manner"

En este momento, nuestro ahora no tan feliz programador, decide realizar un diagrama de clases para empezar a acomodar sus ideas. "Quizás soy yo el problema" - pensó.

El siguiente diagrama de clases muestra el modelo de representación de fechas de Java



Class Date (java.util.Date)

(NOTA: El diagrama está incompleto para mantener claridad.)

Al leer la documentación para realizar este diagrama, lo primero que le llamó la atención fue que hayan tantos métodos **deprecated**. Veamos alguna que **no** esté marcada como **deprecated**, por ej:

void setTime(long time)
Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT.

"Bien!!" - pensó - "Entonces a un objeto de la clase Date podemos setearle el tiempo enviándole un mensaje con un parámetro de tipo long (¿?) que son los milisegundos … ¿queeeeeeeeeeeeeeeé?"

Si este es un método no **deprecated** entonces no quería imaginar uno que lo fuese. Su curiosidad le ganó y leyó lo siguiente:

int getDate()
Deprecated. As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY_OF_MONTH)*.

void setDate(int date)
Deprecated. As of JDK version 1.1, replaced by Calendar.set(Calendar.DAY_OF_MONTH, int date)+.

5. ¿Le parece correcto que la clase **Date** pueda recibir un mensaje **getDate()** que retorna un **int**? ¿Qué espera que retorne este mensaje? (ver *)
6. La clase sabe responder a **setDate(int date)**. ¿Qué opina sobre esto? (ver +)
7. ¿El nombre **Date** de la clase tiene el mismo significado que el parámetro **date** del mensaje **setDate?**

En este punto, nuestro amigo, no sabía si estaba leyendo la documentación del modelo de representación de fechas de Java o un libro de terror. ¿Qué pasos debía seguir? Quizás cometió un error al haber leído los métodos antes de leer los constructores de la clase (algo que siempre hacía) y ahí radicaba todo el problema.

En el diagrama no se muestran los constructores de esta clase. Aquí van, con la documentación correspondiente:

Constructor Summary
Date() Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
Date(int year, int month, int date) **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date) or GregorianCalendar(year + 1900, month, date).*
Date(int year, int month, int date, int hrs, int min) **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min) or GregorianCalendar(year + 1900, month, date, hrs, min).*
Date(int year, int month, int date, int hrs, int min, int sec) **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min, sec) or GregorianCalendar(year + 1900, month, date, hrs, min, sec).*
Date(long date) Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.
Date(String s) **Deprecated.** *As of JDK version 1.1, replaced by DateFormat.parse(String s).*

Luego de leer esto, pensó:

"Para empezar quisiera poder representar una fecha particular, por ej: 20/08/2010. Aunque pensándolo mejor también me gustaría poder representar una fecha con una hora particular, por ej: 20/08/2010 a las 2:00:00hs. ¿Y por qué no, un año en particular, por ej: Año 2004? wow!! me estoy poniendo ambicioso!! Mejor empiezo por lo primero:

¿Cómo represento una fecha particular, por ej: 20/08/2010?"

Existe un constructor que parece el apropiado:

Date(int year, int month, int date)
Deprecated. *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date) or GregorianCalendar(year + 1900, month, date).*

Como nuestro amigo programador es una persona muy responsable (y curiosa), ante un constructor **deprecated**, decidió seguir la investigación del tema "Date Fiasco" (así decidió llamarlo) leyendo la siguiente documentación:

Calendar:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Calendar.html

Y **GregorianCalendar**:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/GregorianCalendar.html

8. ¿Tiene sentido que el objeto, para representar una fecha, deba ser una instancia de una clase llamada **Calendar**? ¿Qué opinión tiene con respecto al nombre de la clase?
9. Lea la documentación de **Calendar** y **GregorianCalendar**. ¿Le parecen buenas representaciones de fechas? ¿Pueden representar el mes Julio?
10. Con las clases **Calendar** y **GregorianCalendar** aparece una nueva clase llamada **Locale**. ¿Qué problema intenta solucionar? ¿Qué ente de la realidad representa? ¿Sigue siendo el mismo dominio del problema de las fechas?

Existe otro modelo de representación del tiempo para Java llamado: Joda Time (http://joda-time.sourceforge.net/).

11. Compare este modelo con el anterior.
12. Con este modelo ¿podemos ...
    … representar una fecha particular, por ej: 20/08/2010?
    … representar una fecha con una hora particular, por ej: 20/08/2010 a las 2:00:00hs?
    … representar un año en particular, por ej: Año 2004?
    … representar un mes en particular, por ej: Enero?
    … intervalos de tiempo, por ej: de Julio de 2006 hasta Enero de 2009?
13. También quisiéramos poder ejecutar el siguiente comportamiento:
    - Dada dos fechas cuál está antes o después.
    - Mover una fecha hacia adelante o hacia atrás en la línea de tiempo (por ej. Moverse 3 días hacia adelante a partir de una fecha dada).

    ¿Podemos lograrlo con Joda Time?
14. ¿Java es el único lenguaje "moderno" con este problema?
    - Php es el lenguaje más utilizado en servidores web. Observe como modela las fechas http://php.net/manual/es/function.date.php

    Objective-C es el lenguaje más utilizado en dispositivos móviles de ultima generación (junto con Java ya visto más arriba)

https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSDate_Class/

15. **(Para hacer luego de haber completado las clases teóricas de la materia)** Luego de haber analizado los modelos anteriores, proponga un modelo propio para la representación de fechas (concentrándose solo en los items del punto 10)

Ejercicio 3 – Herencia vs Forwarding. Entidades de Software Abiertas para Extensión pero Cerradas para Modificación.

a Modele lo siguiente utilizando herencia: "*Una ventana, como un rectángulo, debe ser capaz de calcular su área*".

b ¿Qué problema encuentra con su diseño del punto a)? Analice qué ocurre si se agrega el siguiente enunciado a la especificación: "*En una nueva versión del S.O. las ventanas pueden ser también circulares*".

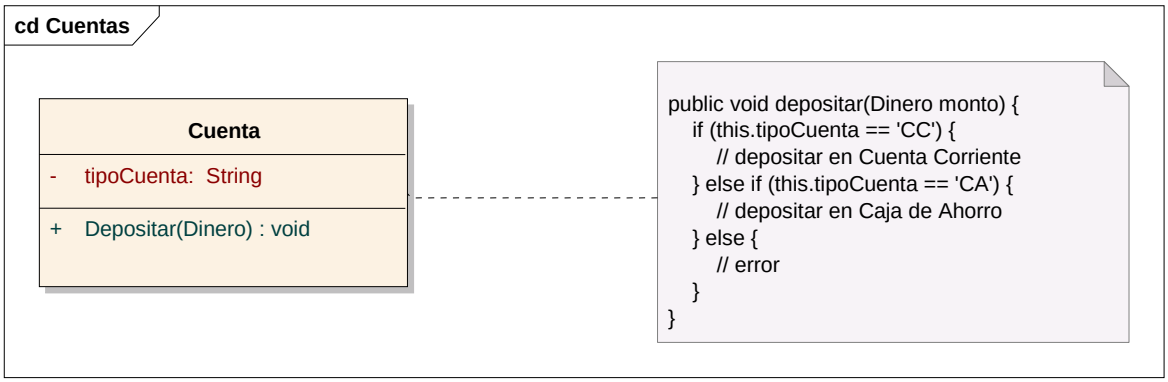
c Modele la misma situación (primero para a) y luego para b)) utilizando composición (forwarding).

d Analice ventajas y desventajas de utilizar herencia vs. utilizar forwarding al diseñar.

Ejercicio 4 – Esencia. Entidades de Software Abiertas para Extensión pero Cerradas para Modificación.

Se cuenta con el siguiente modelo de las cuentas de un banco. Las *Cuentas*, en principio, pueden ser de dos tipos: *Cuenta Corriente* o *Caja de Ahorro.*

(Ignore los signos "+" y "-" de los diagramas).



a) Realice las modificaciones necesarias para incluir el nuevo tipo de cuenta, *Cuenta Corriente Especial.*

b) Bajo las condiciones establecidas, ¿qué desventajas le ve al modelo?

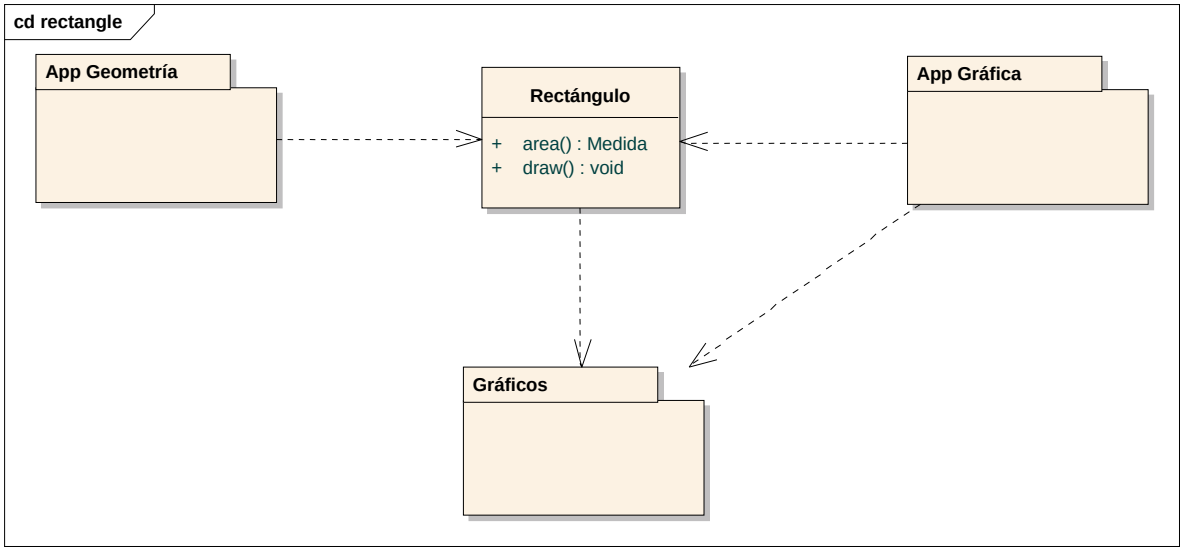
c) En base a sus críticas, proponga un modelo alternativo que solucione dichas desventajas.

Ejercicio 5 – Única Responsabilidad = Eje de Cambio. Alta Cohesión.

Se tienen dos aplicaciones que utilizan una misma clase Rectángulo. Una aplicación es puramente visual (AppGráfica, solo interesa mostrar rectángulos por pantalla), y la otra aplicación sólo realiza cálculos (AppGeométrica).

La clase Rectángulo implementa dos métodos: uno para calcular el área y otro para dibujarse en pantalla.

(Ignore los signos "+" de los diagramas).

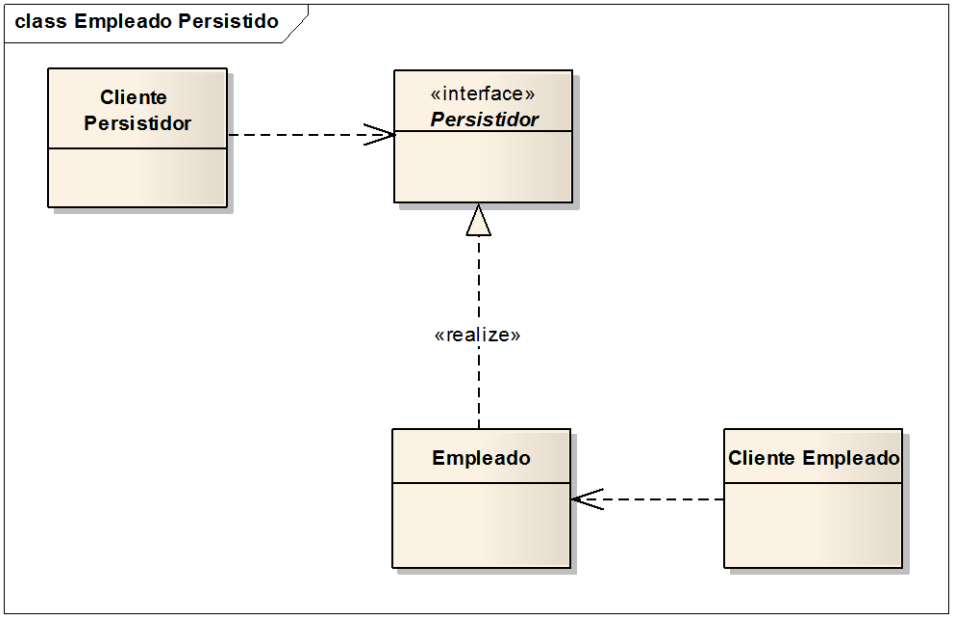


a) ¿Qué problemas encuentra al enfoque propuesto? Piense acerca de cómo ven al Rectángulo ambos tipos de aplicaciones.

b) Proponga un diseño alternativo

Ejercicio 6 - Única Responsabilidad = Eje de Cambio. Alta Cohesión.

i) Se tiene una clase Empleado que implementa una interfaz Persistidor que le permite persistirse (guardarse en cierto formato) en disco. También existe un cliente que se encarga de utilizar dicha implementación, y otro que sólo se encarga de utilizar a los Empleados.



a) ¿Nota algún problema? Piense en términos de cambios de dominio de negocio y cambios de base (por ejemplo, cambios en el método de persistencia en disco).

b) Proponga un diseño alternativo que lo solucione.

ii) Dada la siguiente definición de la clase "Modem"…

```
public class Modem{
      public void llamar(string numeroDeTelefono){…}
      public void cortar(){…}
      public void enviar(char unCaracter){…}
      public char recibir(){…}
}
```

a) Analice la cohesión de la misma teniendo en cuenta que existen en el sistema clases que la utilizan sólo con respecto a la administración de conexiones y otras que lo hacen únicamente para el pasaje de mensajes.

b) ¿Cómo hubiese sido su análisis de no conocer la información provista en a)?

Ejercicio 7 - Inversión de Dependencias. Bajo Acoplamiento.

Supongamos que se necesita leer caracteres de cierto dispositivo *A*, traducirlos de alguna forma y escribirlos en otro dispositivo *B*.

Suponiendo un lenguaje de programación de chequeo de tipos estático, realizamos un primer diseño que soluciona el problema mediante el siguiente enfoque:

```
public class TranslatorAToB{
[…]
      ReaderA reader;
      WriterB writer;
      public translate(){
            while (!this.reader.end()){
                  char unCaracter=this.reader.read();
                  // traducir unCaracter.
                  this.writer.write(unCaracterTraducido);
            }
      }
[…]
}
public class ReaderA{
[…]
      public char read(){…}
      public bool end(){…}
[…]
```

```
}
public class WriterB{
[…]
      public void write(char unCaracter){…}
[…]
}
```

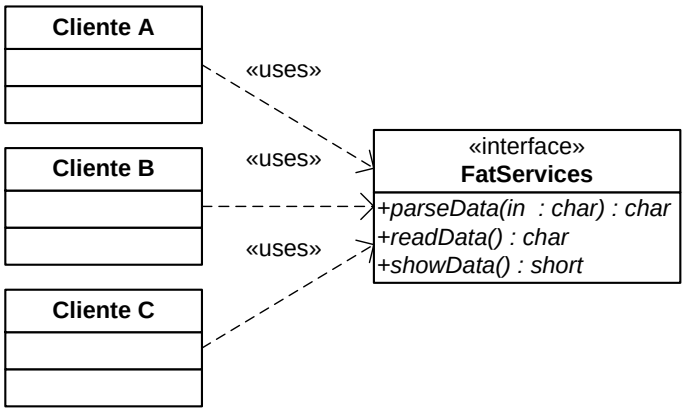
a) ¿Qué ocurre si se nos presenta un nuevo dispositivo de lectura *C* o uno de escritura *D*?

b) Realice un modelo alternativo que solucione este problema.

c) ¿Se tiene este problema en un lenguaje de chequeo de tipos dinámico? ¿Por qué?

Ejercicio 8 – No depender de Mensajes que no se utilizan.

Asumiendo nuevamente que estamos trabajando con un lenguaje de chequeo de tipos estático, contamos ahora con una interfaz con muchos métodos (*fat interface*) que es consumida por muchos clientes:



(Ignore los signos "+" de los diagramas).

Instancias de ClienteA sólo envían el mensaje showData(), instancias de ClienteB sólo envían parseDate() e instancias de ClienteC sólo readData().

Al tiempo de haber publicado nuestra interfaz FatServices, instancias del ClienteC además de leer datos necesitan un servicio para validarlos, entonces el programador de ClienteC pide implementar un nuevo mensaje en FatServices llamado validateData().

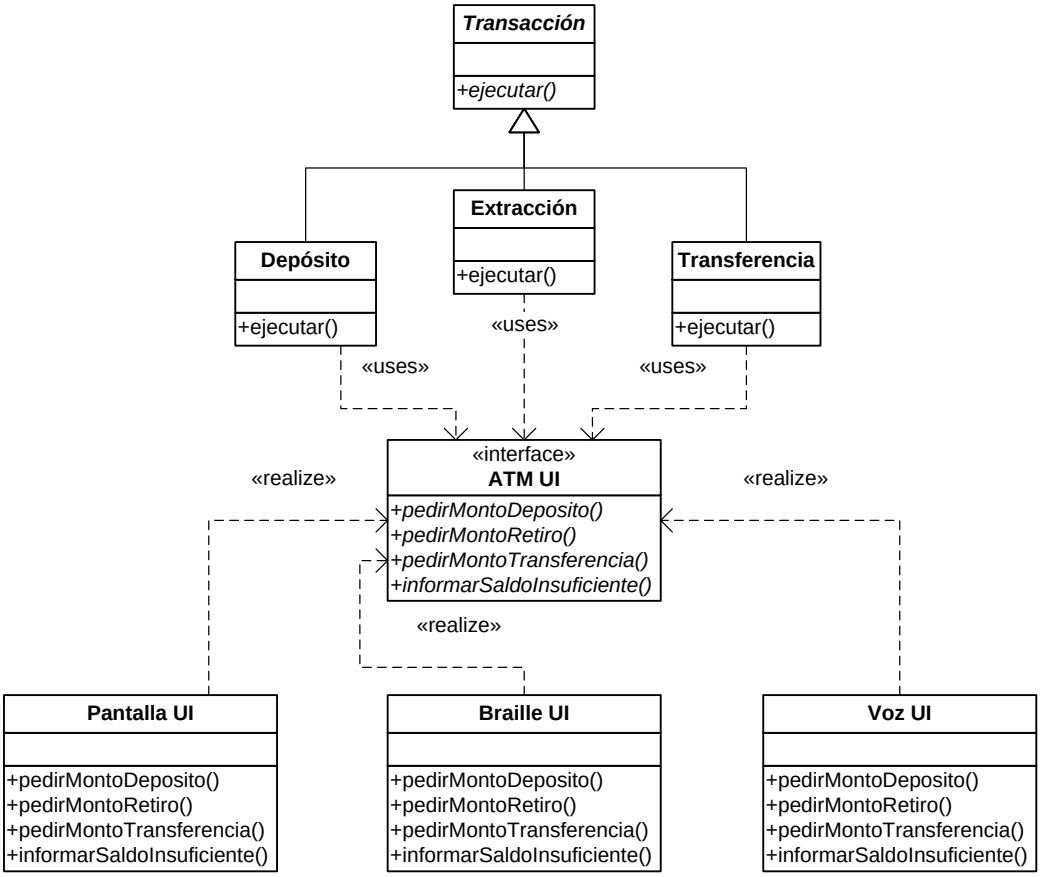
a) ¿Qué problemas encuentra al enfoque propuesto? Piense en los otros Clientes.

b) Proponga un diseño alternativo.

Ejercicio 9 - No depender de Mensajes que no se utilizan.

El siguiente diagrama de clases muestra el funcionamiento de un típico cajero automático (en ingles ATM). La interfaz del mismo debe ser muy flexible, y por ej. poseer la capacidad de interactuar por voz, con una tabla de braille, o por medio de la pantalla y en múltiples idiomas. Por otro lado, cada posible transacción del cajero

es encapsulada en una subclase de la clase abstracta "*Transacción*" y utiliza los mensajes provistos por esta según lo necesite cada transacción.



(Ignore los signos "+" de los diagramas).

Se pide:

a) Asumiendo un lenguaje con chequeo estático de tipos, ¿Qué problema encuentra con este diseño?

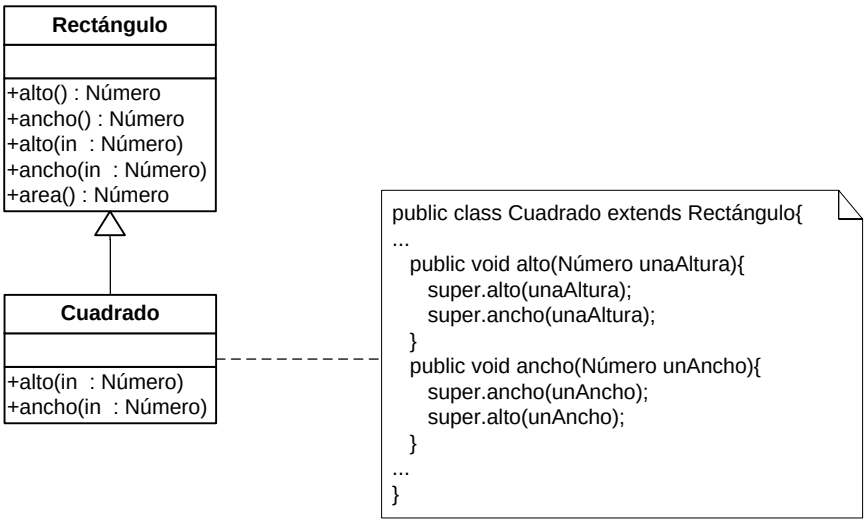
b) ¿Cómo podría solucionarlo?

c) ¿Se tiene este problema en un lenguaje con chequeo dinámico de tipos?

Ejercicio 10 – Clases hijas se comporten como sus padres.

Asumiendo el contexto de una aplicación gráfica/geométrica en la que **se necesita modificar el tamaño de figuras con mucha frecuencia**, considerar la clase Rectángulo y la clase Cuadrado definidas en el diagrama de más abajo. Desde un punto de vista matemático, un cuadrado es un caso particular de un rectángulo, con lo cual, un posible enfoque sería el siguiente:

(Ignore los signos "+" de los diagramas).



a) ¿Qué opina de este modelo?

b) Ahora analicemos el siguiente código:

```
public class ProbarRectangulo {

      public static void prueba(Rectangulo unRectangulo) {
            unRectangulo.ancho(4);
            unRectangulo.alto(5);
            System.out.println("Ancho = 4, Alto = 5, Area = " +
            unRectangulo.area());

            if (unRectangulo.area() == 20)
                  System.out.println("OK!\n");
            else
                  System.out.println("Algo anda mal.\n");
      }

      public static void main(String args[]) {
            Rectangulo unRectangulo = new Rectangulo(1, 1);
            Cuadrado unCuadrado = new Cuadrado(1);
            prueba(unRectangulo);
            prueba(unCuadrado);
      }
}
```

```
Cuya salida es la siguiente:

Ancho = 4, Alto = 5, Area = 20
OK!
Ancho = 4, Alto = 5, Area = 25
Algo anda mal.
```

¿Qué está ocurriendo? ¿Cuál es el problema? ¿Cuál es la definición de herencia en diseño orientado a objetos?

c) ¿Cómo solucionaría el problema?

Ejercicio 11 - Clases hijas se comporten como sus padres.

i) Tenemos la siguiente especificación en lenguaje natural:

”Un empleado puede trabajar mensualmente y recibir un sueldo fijo, o trabajar por horas y recibir un sueldo en función a sus horas trabajadas obtenidas a partir de la suma de sus tarjetas de hora semanales”.

a) Haga un diagrama de clases a partir de la especificación para liquidar sueldos de empleados.

b) Se agrega lo siguiente a la descripción: “Adicionalmente existen voluntarios que lo hacen sin percibir dinero”. Complete su modelo de a)

c) Analice y critique su propio modelo efectuado en b). ¿Ve algún problema? ¿Cómo lo solucionaría?

Ejercicio 12 - Clases hijas se comporten como sus padres.

Un desarrollador está trabajando en un paquete de funciones matemáticas / geométricas. En el transcurso de su trabajo escribe la siguiente clase para representar rectas (descartando las verticales):

```
public class Recta {
      private Punto unPunto;
      private Punto otroPunto;
      public Recta(Punto unPunto, Punto otroPunto)
      { this.unPunto=unPunto; this.otroPunto=otroPunto; }
      public Punto punto1() { return unPunto; }
      public Punto punto2() { return otroPunto; }
      public double pendiente() { /*código*/ }
      public Punto interseccionEjeY() { /*código*/ }
      public boolean pertenece(Punto p) { /*código*/ }
}
```

Luego tiene la tarea de escribir una clase para representar segmentos de recta (tampoco le interesan los segmentos verticales). Como encuentra que un segmento es en realidad una recta “finita” y comparte funcionalidad con la recta, decide escribir lo siguiente para aprovechar su trabajo previo:

```
public class Segmento extends Recta {
      public Segmento (Punto unPunto, Punto otroPunto){ super(unPunto,
      otroPunto); }
      public double longitud() { /*código*/ }
      public boolean pertenece(Punto p) { /*código*/ }
}
```

a) ¿Qué problema encuentra con este diseño?

b) ¿Cómo podría solucionarlo?

Parte III:

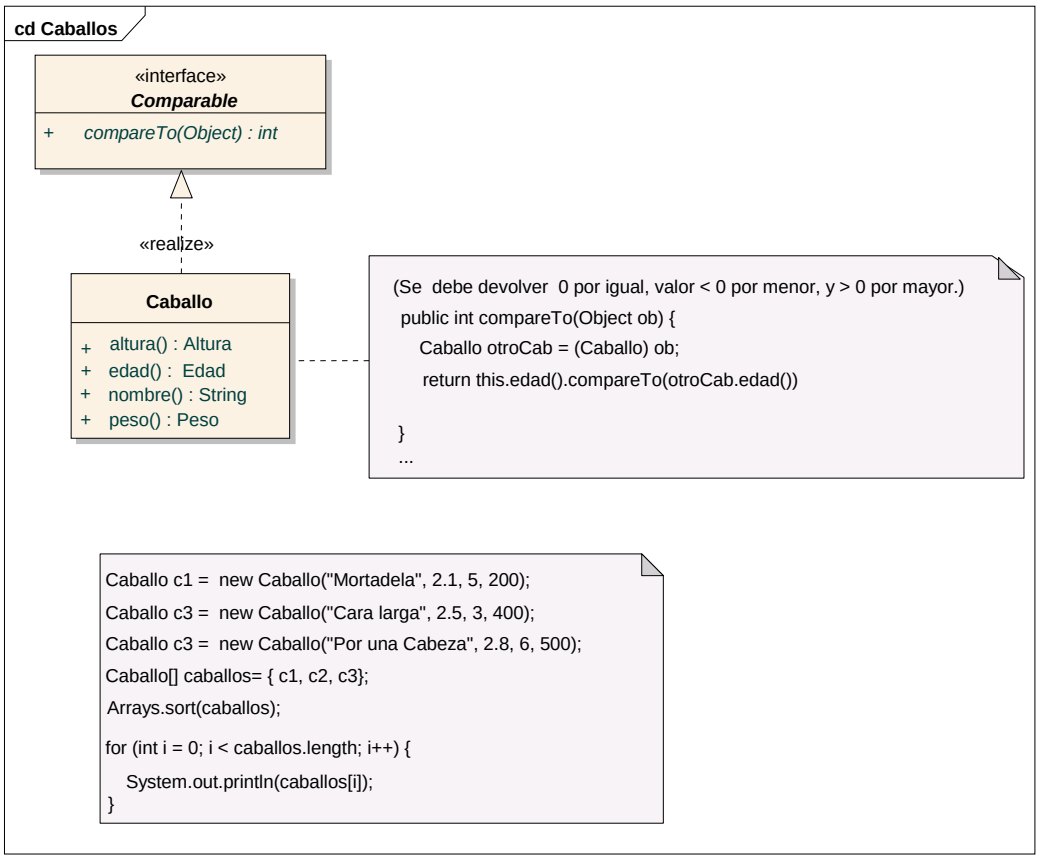
Modelado. Patrones de diseño simples. Diseño iterativo incremental. Integración partes 1 y 2. Idioms. Closures. Metaprogramación. Double Dispatch. Eliminación de "if" utilizando polimorfismo. Patrones de Diseño más Complejos.

Ejercicio 1 - Modelado. Faltan "objetos".

Eustaquio el burrero (un amigo de la casa), nos pidió un pequeño sistema que le permitiera comparar distintos caballos para poder determinar a cuál debía jugarle. En principio nos dijo que una estrategia que le funcionaba, era apostar a los caballos más jóvenes.

Luego de pensar unos minutos, le entregamos la siguiente solución:

(Ignore los signos "+" de los diagramas).



Arrays.sort() es un método de la clase Arrays que permite ordenar arrays de objetos que implementen la interfaz Comparable.

Eustaquio probó la solución, y se fue más que contento.

Una semana después, nuestro querido amigo volvió a charlar con nosotros, su táctica de apostar al más joven no estaba funcionando. Quería ahora, además de poder ordenar por edad, ordenar por peso (había algunos caballitos jóvenes que estaban siendo muy bien alimentados y estaban un poco rechonchos).

Ningún problema dijimos, es muy simple… ¿o no?

a) ¿Cómo se le ocurre solucionar el problema que nos plantea Eustaquio?

b) ¿Cuánto le "cuesta" a su solución propuesta en a) permitir comparar también caballos por altura?

c) Presente una nueva solución que permita no sólo comparar por edad o peso según Eustaquio requiera, sino que frente a futuros pedidos de este estilo por parte de Eustaquio, el impacto del cambio sea despreciable.

Ejercicio 2 – Modelado. Reinventar la Rueda. Patrones simples.

Se desea implementar el código controlador de una máquina expendedora de chicles.

Alguien escribió las siguientes líneas de código para solucionar el problema:

```
public class MaquinaDeChicles{
      final static int AGOTADO=0;
      final static int SIN_MONEDA=1;
      final static int CON_MONEDA=2;
      final static int VENDIDO=3;

      int estado = VENDIDO;
      int cantidad = 0;

      public MaquinaDeChicles(int cant){
            this.cantidad = cant;
            if (cantidad > 0)
                  estado = SIN_MONEDA;
      }
      public void insertarMoneda(){
            if (estado == CON_MONEDA){
                  System.out.println("No podes insertar otra moneda");
            }else if (estado == SIN_MONEDA){
                  estado=CON_MONEDA;
                  System.out.println("Insertaste una moneda");
            }else if (estado == AGOTADO){
                  System.out.println("No quedan más chicles…");
            }else if (estado == VENDIDO){
                  System.out.println("Espera a que salga tu chicle
                  primero!");
            }
      }
      public void ejectarMoneda(){
            if (estado == CON_MONEDA){
                  System.out.println("Moneda devuelta");
                  estado=SIN_MONEDA;
            }else if (estado == SIN_MONEDA || estado == AGOTADO){
```

```
                System.out.println("No insertaste ninguna moneda");
          }else if (estado == VENDIDO){
                System.out.println("Lo siento, ya giraste la
    palanca").
          }
    }
    public void girarPalanca(){
          if (estado == VENDIDO){
                System.out.println("Girar dos veces no te va a dar
                más chicles!");
          }else if (estado == SIN_MONEDA){
                System.out.println("Giraste, pero no hay monedas…");
          }else if (estado == AGOTADO){
                System.out.println("Giraste, pero no hay chicles…");
          }else if (estado == CON_MONEDA{
                System.out.println("Giraste la palanca y…");
                estado=VENDIDO;
                dispensar();
          }
    }
    public void dispensar(){
          if (estado == VENDIDO){
                System.out.println("… aquí está tu chicle!");
                cantidad --;
                if (cantidad == 0){
                      estado = AGOTADO;
                }else{
                      estado = SIN_MONEDA;
                }
          }else if (estado == SIN_MONEDA){
                System.out.println("Error, no debería ocurrir");
          }else if (estado == AGOTADO){
                System.out.println("Error, no debería ocurrir");
          }else if (estado == CON_MONEDA{
                System.out.println("Error, no debería ocurrir");
          }
    }
}
[Otros métodos como agregarChicles…]
```

a) Describa el diagrama de clases del código de arriba y critique el diseño de esta solución.

b) Tenemos el siguiente requerimiento de cambio: como promoción especial, queremos que 1 de cada 10 veces las expendedoras devuelvan dos chicles en vez de sólo uno. ¿Cómo impacta esto en el diseño de la solución presentada?

c) Generar un diagrama de clases de un diseño alternativo que evite los problemas del punto anterior y que tenga en cuenta sus criticas del punto a).

Ejercicio 3 – Modelado. Reinventar la Rueda. Patrones simples.

Una persona quiere realizar un viaje a un destino determinado. Para hacerlo tiene varias alternativas, por ejemplo, tomar un taxi, un colectivo, un tren, ir en su propia moto o a caballo.

Antes de tomar la decisión tiene en cuenta todas las alternativas posibles y, para decidirse por una, utiliza como criterio cuánto dinero tiene disponible en ese momento, pero no se descarta que en el futuro el criterio se modifique.

a) Ejemplifique **todo** el proceso a través de escenarios y diagramas de secuencias.
b) Construya un diagrama de clases para la situación que contemple los posibles futuros cambios especificados.

Ejercicio 4 - Modelado. Reinventar la Rueda. Patrones simples.

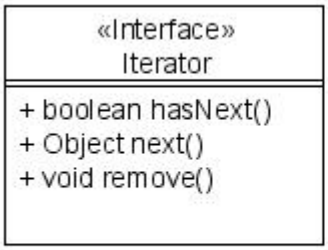
Se quiere diseñar una rocola (jukebox). Para que esta funcione, se requiere que se le ingrese primero una ficha. Luego, el usuario puede elegir ordenar los temas de alguna manera (por orden alfabético de los nombres de los temas, o cantautor, o nombre del álbum, etc.) y presionar *next* o *prev* para recorrer la lista de canciones obtenidas. Cuando la persona finalmente encuentra el tema a escuchar presiona *play* para que éste comience a ser reproducido. Al terminar la canción, deberá colocar una nueva ficha si quiere repetir el proceso.

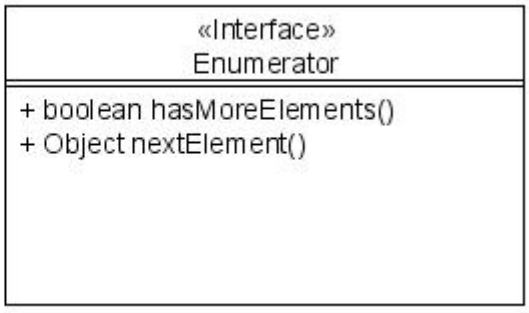
Realice un diagrama de clases y secuencias que describa la situación.

Ejercicio 5 - Modelado. Reinventar la Rueda. Patrones simples.

En Java existen las siguientes interfaces: Iterator y Enumeration. Enumeration es una interfaz antigua y actualmente se utiliza más Iterator.

(Ignore los signos "+" de los diagramas).





Es común que suceda que utilizando algún *framework* o biblioteca llamamos a un método y obtengamos un Iterator (o una Collection y podemos pedirle un Iterator).

a) Diseñe una solución para seguir utilizando los mensajes que reciben un Enumerator.

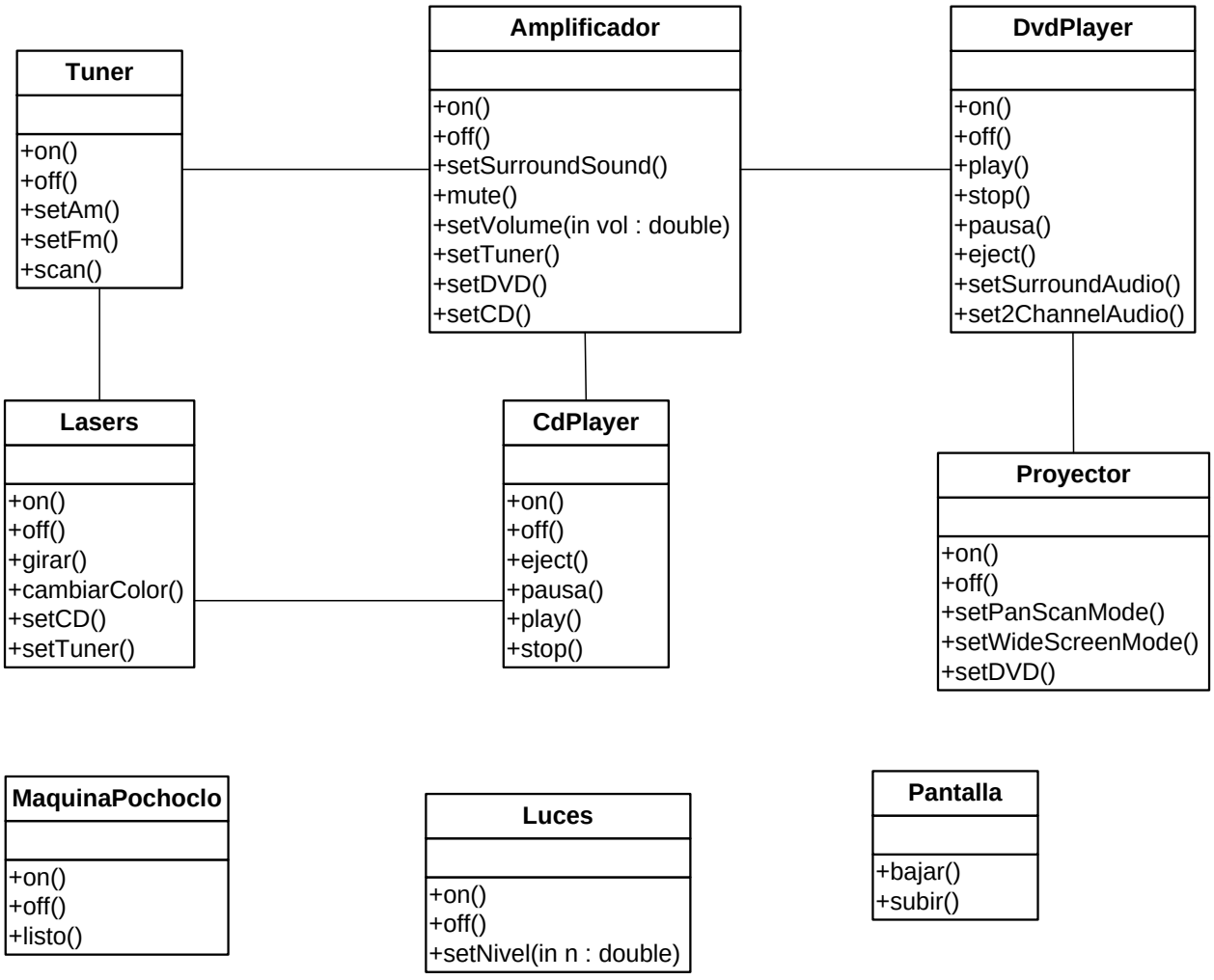
b) ¿Y si tenemos el problema inverso? Si se posee un mensaje que recibe un Iterator y nosotros tenemos un Enumerator, ¿De qué forma podemos seguir utilizando código que recibe un Iterator? Diseñe y escriba el pseudocódigo con una solución. ¿Su solución quiebra algún principio de diseño?

Hint: la documentación de la interfaz Iterator *indica que el método* remove*() puede lanzar* UnsupportedOperationException *si el método no es soportado por la implementación particular del iterador.*

Ejercicio 6 - Modelado. Reinventar la Rueda. Patrones simples.

Se tienen los siguientes componentes en el diseño de un salón de usos múltiples:

(Ignore los signos "+" de los diagramas).



Suponga ahora un objeto cliente que quiera hacer uso de las funcionalidades de los objetos que son instancias de las clases declaradas, por ejemplo un "*Home Pochocling Theater"*. En ese contexto, el proceso correcto para ver una película es:

1) Prender la máquina de pochoclo.

2) Bajar la luces a nivel a lo sumo 3.
3) Bajar la pantalla.
4) Prender el proyector.
5) Colocar el input del proyector en DVD.
6) Colocar al proyector en modo *widescreen*.
7) Prender el amplificador de sonido.
8) Colocar el input del amplificador en DVD.
9) Colocar el amplificador en modo *Surround Sound*.
10) Colocar el amplificador en volumen 5 (medio)
11) Prender el DVD.
12) Esperar a que la máquina de pochoclo tenga listos los pochoclos.
13) Presionar *play* en el DVD.

a) Analizar que ocurre por cada cliente que quiera hacer uso de alguna de las funcionalidades de este subsistema (un ejemplo de uso es el comenzar a ver una película descripto arriba. Otro podría ser el finalizar de verla). ¿Y si se modifica/mejora algún componente?

b) Proponga una solución para estos problemas. Suponga que todos los clientes van a ver al subsistema como un "*Home Pochocling Theater*".

c) Ahora suponga que otros clientes desean usar los componentes para armar una discoteca hogareña (con el CD player, los láseres, el amplificador, etc…) ¿En que impacta esto en su solución anterior?

Ejercicio 7 - Modelado. Reinventar la Rueda. Patrones simples.

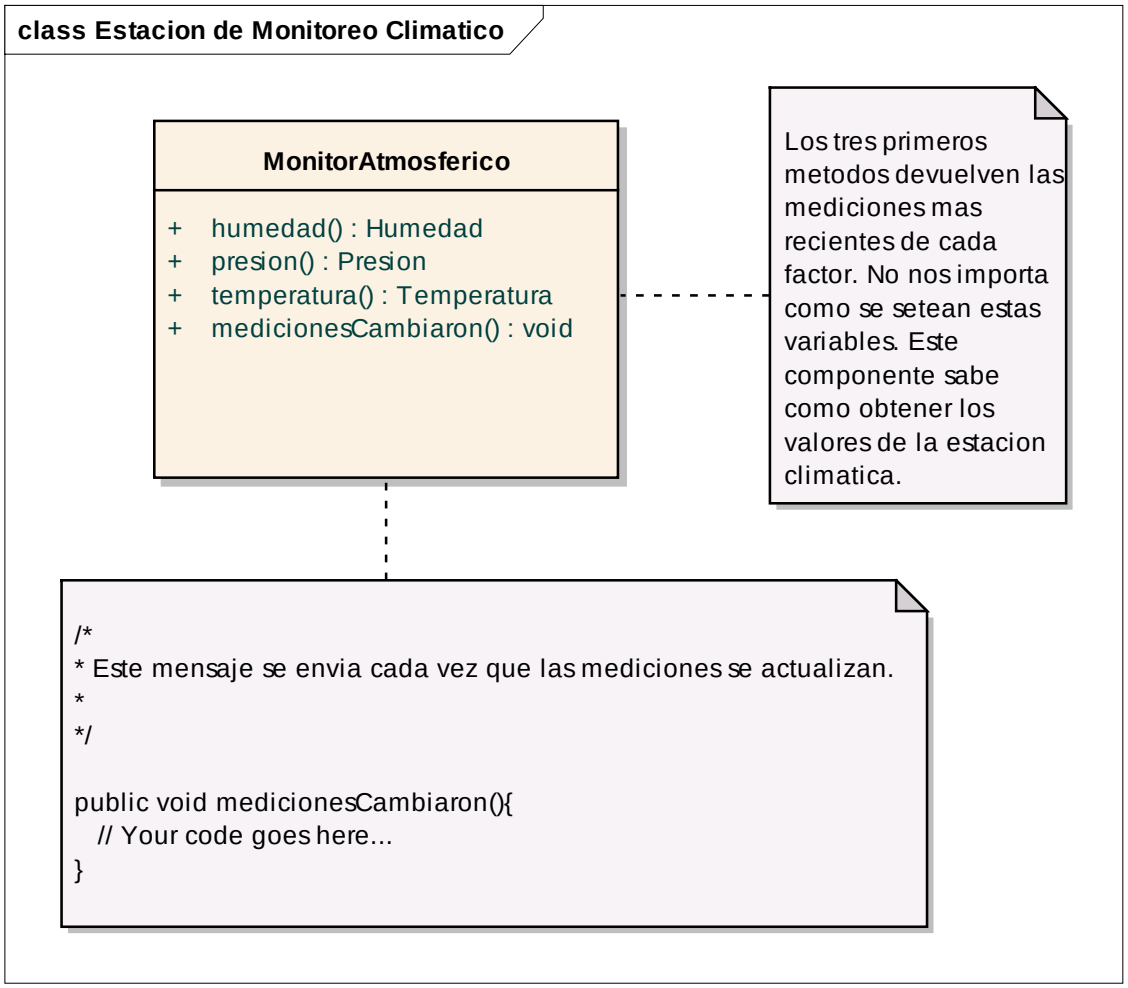
Nos contratan para diseñar un componente encargado de monitorear las condiciones meteorológicas (temperatura, humedad y presión barométrica) registradas por una estación climática (dispositivo físico capaz de medir parámetros del clima).

La aplicación debería proveer inicialmente de tres displays diferentes para mostrar tres aspectos distintos: condiciones actuales, estadísticas climáticas sobre período de tiempo y pronóstico para los próximos días, todos actualizados en tiempo real mientras la estación climática obtiene las últimas y más recientes mediciones.

Se desea que el componente sea fácilmente extensible para que en el futuro otros desarrolladores puedan escribir sus propios displays en base a la información medida.

Nos proveen de la siguiente componente para efectuar nuestro trabajo:

(Ignore los signos "+" de los diagramas).



Un compañero de trabajo nos propone entonces el siguiente código:

```
public class MonitorAtmosferico{

[…]
      public void medicionesCambiaron(){
            Temperatura temp = temperatura();
            Humedad hum = humedad();
            Presion pre = presion();

            displayCondicionesActuales.update(temp, hum, pre);
            displayEstadisticas.update(temp, hum, pre);
            displayPronostico.update(temp, hum, pre);
      }
[…]
}
```

a) Haga un diagrama de clases de la versión propuesta por el compañero de trabajo.

b) Critique dicho diseño.

c) Confeccione un diagrama de clases de una mejor solución. Justifique.

d) Haga un diagrama de secuencias completo de la invocación al método medicionesCambiaron() de su solución del punto c).

Ejercicio 8 – Double Dispatch. Eliminación de "if".

Suponga que tiene el siguiente extracto de código:

```
[…]
if (unNumero == 0) {
      throw unaExcepcion;
}
[…]
```

Reescriba el código evitando usar IF.

Hint 1: Piense un objeto (por ej: unaAserción) que sabe responder dos mensajes:

1. `recibirUnCero`
2. `recibirUnNoCero`

¿Qué está representando el objeto unaAserción?

¿Qué realizará este objeto cuando se le envía el mensaje recibirUnCero?

Hint 2: Suponga que tiene dos objetos: Cero y Diez que representan al número 0 y al número 10 respectivamente. Ahora suponga que tienen un mensaje

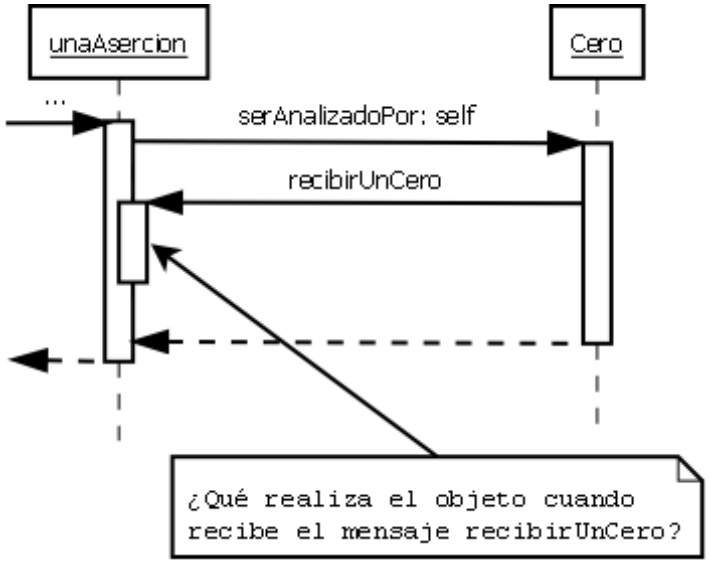
`serAnalizadoPor: unaAserción`

con respecto al cual estos dos objetos son polimórficos. ¿Qué hará el objeto Cero al recibir el mensaje serAnalizadoPor: unaAserción? (rever los dos mensajes en **Hint1**)

Hint 3: Vea el patrón de diseño Visitor, y **en particular** la técnica de double dispatch que este utiliza.

Hint 4: ¿Todavía no se le ocurre? … una posible solución necesita los siguientes objetos: unaAsercion, Cero, Diez y unaExcepcion.

Hint 5: ¿Esto ayuda?



Ejercicio 9 – Modelado. Manejo de errores. Idioms. Closures.

Supongamos que hay un objeto que representa un conjunto de clientes que sabe responder el mensaje *#clienteLlamado*: *aName.* Este mensaje devuelve un cliente cuyo nombre es *aName*, pero si no lo encuentra devuelve el objeto **nil**. Realice un análisis crítico de esta decisión de diseño (retornar **nil** si no se encuentra el cliente) y proponga una mejor solución si cree usted que existe.

Ejercicio 10 – Código Repetido. Idioms. Closures.

A partir del siguiente pseudocódigo:

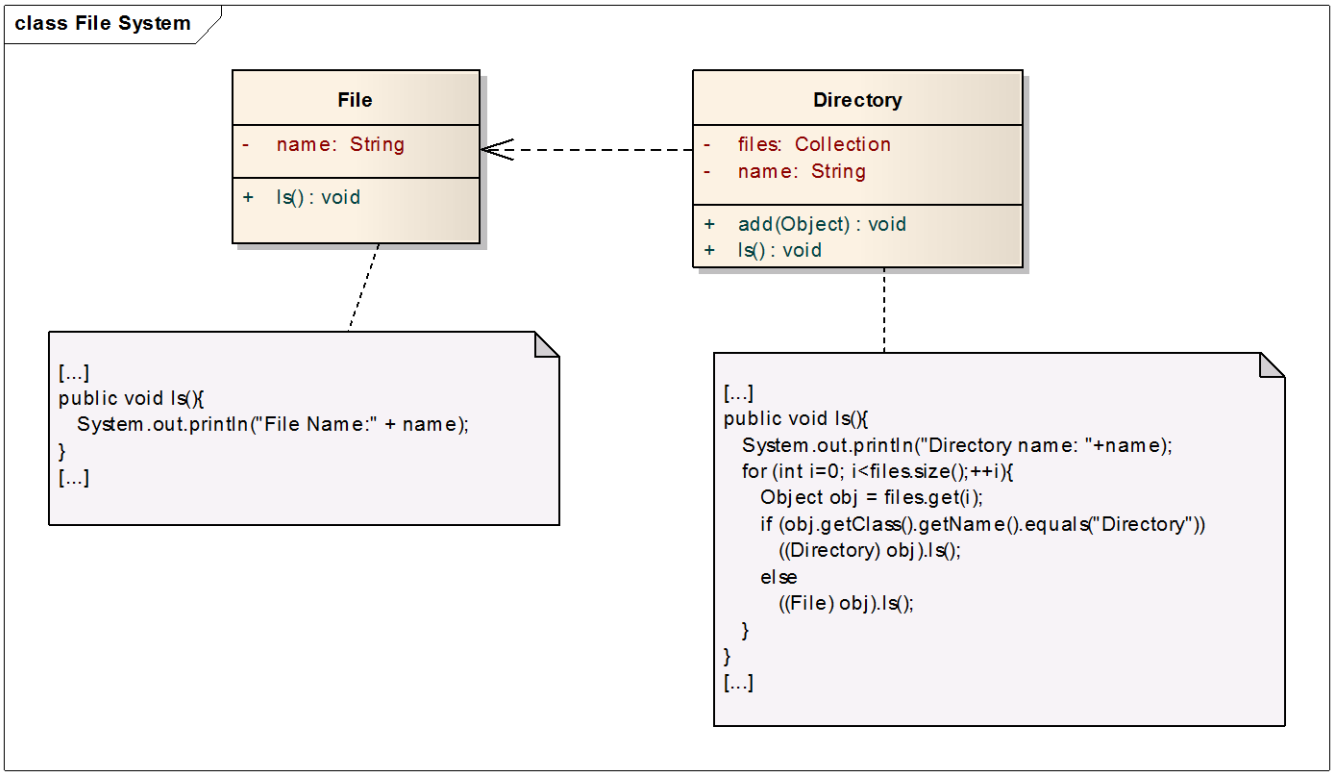
```
Class Listador {
        [...]
        listaSalarios ListarSalariosMayoresDe: monto En: empleados {
                salariosMayores = MutableArray New;
                Foreach(empleado on empleados) {
                        salario = empleado Salario;
                        If (salario > monto) {
                                salariosMayores Add: salario;
                        }
                }
                return salariosMayores;
        }
        [...]
}
[...]
        void Prueba {
                […] -> // Se crea y utiliza empleados…
```

```
            Try {
                unListador = Listador New;
                salariosMayores = unListador ListarSalariosMayoresDe: $1000
                En: empleados;
            } Catch (Exception ex){
                System Log: "Error listando salarios mayores a cierto monto";
            }
            [...]
        }
[...]
```

Juzgue al pseudocódigo presentado. De su crítica ser negativa, proporcione una versión alternativa superadora.

Ejercicio 11 - Modelado. Reinventar la Rueda. Patrones complejos.

Alguien dejó el siguiente diseño de un *File System*:



El mismo supuestamente permite modelar un sistema de directorios con sus archivos, y pedir la impresión de dicha lista por pantalla (ls())

El siguiente código ejemplifica su uso:

```
public static void main(String[] args) {
        Directory myDoc = new Directory("My Documents");
        Directory pictures = new Directory("Pictures");
        Directory music = new Directory("MP3");
```

```
        File f1 = new File("Nicole.jpg");
        File f2 = new File("Pampita.jpg");
        pictures.add(f1);
        pictures.add(f2);
        myDoc.add(pictures);

        File f3 = new File("Zapato Veloz – Tengo un tractor
                                amarillo.mp3");
        music.add(f3);
        myDoc.add(music);
        myDoc.ls();
}
El resultado esperado es:
Directory name: My Documents
Directory name: Pictures
File name: Nicole.jpg
File name: Pampita.jpg
Directory name: MP3
File name: Zapato Veloz – Tengo un tractor amarillo.mp3
```

a) ¿Qué problemas encuentra en el diseño propuesto? Tenga en cuenta que los siguientes son **algunos** de los posibles escenarios de cambio:

i) ls() puede ser alterado para que provea más información acerca de los archivos.

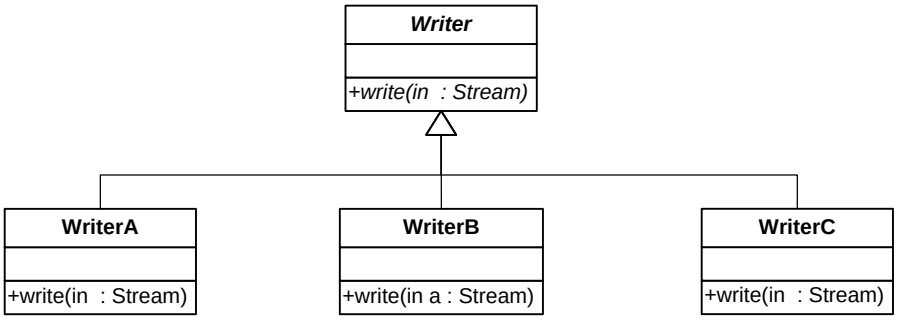
ii) Proveer la posibilidad de crear *links simbólicos* entre archivos y/o directorios.

b) Proponga una solución alternativa

Ejercicio 12 - Modelado. Reinventar la Rueda. Patrones complejos.

Nos presentan el siguiente diseño, que permite la escritura de *streams* de texto en muy diferentes dispositivos (A, B y C). Este esquema se utiliza desde hace un tiempo y se desea no tocarlo, ni modificarlo en lo absoluto. Se sabe además que múltiples objetos "clientes" utilizan a la abstracción *writer* para ejecutar el mensaje *write* para streams que desean escribir, y así no acoplarse a una implementación concreta. Se desea mantener sin cambios dicha forma de operar.

(Ignore los signos "+" de los diagramas).



Sin embargo, súbitamente están apareciendo algunos problemas con las escrituras en los dispositivos, por lo que nos informan que se desea tener la posibilidad de loguear las escritura para encontrar el problema. **Se nos repite que no se quiere modificar ningún código ya existente y que todo sea transparente.**

i) Teniendo en cuenta las restricciones planteadas, extienda el modelo para contemplar la nueva funcionalidad.

ii) ¿Qué opina, si para solucionar el punto i) agrega a la jerarquía las clases LoggedWriterA, LoggedWriterB, y LoggedWriterC? (escribiendo para cada una su implementación propia de *write*).

iii) Encontrado el problema debido a la solución propuesta, se nos pide agregar nuevas funcionalidades. Como los streams están ocupando mucho espacio en los dispositivos en cuestión, se desea que puedan, también de forma transparente, comprimir los streams antes de escribirlos. ¿Cómo impacta esto en su solución anterior?

iv) Con nuestra solución de iii) en mano, nos informan que ahora también se busca que un *writer*, dado un *stream*, pueda encriptar el texto para que no pueda entenderse por usuarios mal intencionados, utilizando algún esquema de encriptación.

v) El problema de i y ii) vuelve a aparecer. Se quiere poder loguear la escritura de un stream que antes es primero encriptado y luego comprimido.
Extienda su modelo de iv) para que contemple los nuevos requerimientos. Efectúe un diagrama de secuencias del mensaje *write* del *writer* descripto en el último ejemplo, utilizando su solución propuesta.

Ejercicio 13 - Modelado. Reinventar la Rueda. Patrones complejos.

i) En base a la Clase abstracta "Boolean" vista en las clases teóricas, con las subclases concretas "True" y "False, efectuar un modelo capaz de representar cualquier expresión booleana, por ej. ¬(trueVfalse). La expresión debería de poder evaluarse para conseguir su valor booleano.

ii) Desarrollar un modelo, capaz de representar una expresión aritmética de suma y que la misma pueda ser realizada (o sea, ejecutada).

iii) Ahora se quiere crear un modelo para las siguientes operaciones binarias: +, -, * y /.

Hint: Considere una estructura de árbol para modelar las operaciones.

Ejercicio 14 - Modelado. Reinventar la Rueda. Patrones complejos.

Adapte su solución del ejercicio 2 i) para poder responder al mensaje "contarOrs" que devuelve la cantidad de "ors" (V) contenida en una expresión booleana específica. También se desea que una expresión booleana sepa imprimirse en un String. Obtener un diseño que permita la extensibilidad de estas, y futuras operaciones a responder por una expresión booleana (por ej. una futura "contarAnds") sin modificación de código, y que mantenga alta la cohesión de la expresión booleana. ¿Puede lograrlo? ¿Con qué problema se encuentra?

Ejercicio 15 –Closures.

La mayoría de los lenguajes modernos de programación permite el uso de clausuras, bloques, funciones de alto orden o funciones anónimas.

El uso más común es para diferir un callback cuando ocurre un evento externo asincrónico.

1) Realice un diseño que permita ejecutar un bloque de código arbitrario cuando finaliza una tarea externa. Dicha tarea se realiza en un servidor y es ejecutada por el cliente de manera asincrónica.

2) Suponga que el bloque de código a ejecutar es enviar un correo electrónico a una dirección de email predeterminada y debe realizarse una vez que se verifica que un servidor está online. Escriba el diagrama de secuencias completo desde que se registra el callback hasta que se ejecuta (una vez terminada la tarea externa). Ignore el caso donde el servidor no responde. ¿Qué dificultades se le plantean?

3) ¿Qué debería modificar de su diseño para contemplar el caso en que se quiere definir un bloque de callback para el caso que el servidor no responde?

Ejercicio 16 –Closures.

Suponga que desea recorrer una estructura arbitraria iterando sus elementos y contando cuantos son.

1) Escriba un bloque de código que vaya acumulando la cantidad visitada.

2) Escriba otro bloque que vaya guardando el orden en que la recorrió y retorne al final una bitácora de los elementos visitados.

3) ¿Qué ventajas tiene utilizar una clausura para efectuar este trabajo?

4) ¿Se le ocurre una implementación alternativa utilizando algún patrón de diseño?

Ejercicio 17 – Metaprogramación.

1. Defina metaprogramación en DOO.
2. A su criterio: ¿Cuáles son los mensajes que una clase debería saber responder?
3. ¿Es correcto que una clase sepa responder al mensaje **new**?
4. Explique el funcionamiento del siguiente código (Java):

```
//Java With reflection
Class cls = Class.forName("Foo");
Object foo = cls.newInstance();
Method method = cls.getMethod("hello", null);
method.invoke(foo, null);
```

5. Haga lo propio con el siguiente bloque de código (Smalltalk):

```
Object subclass: #Account.
 Account instanceVariableNames: 'balance'.

Account comment:
 'I represent a place to deposit and withdraw money'

Account class extend [
```

```
    new [
        | r |
        <category: 'instance creation'>
        r := super new.
        r init.
        ^r
    ]
]

Account extend [
      init [
        <category: 'initialization'>
        balance := 0
    ]
]

Account extend [
        spend: amount [
            <category: 'moving money'>
            balance := balance - amount
        ]
        deposit: amount [
            <category: 'moving money'>
            balance := balance + amount
        ]
]

cuenta1 := Account new.
cuenta1 deposit: 100.
cuenta1 spend: 10.

cuenta2 := Account new.
cuenta2 deposit: 50.
cuenta2 spend: 5.
```

6. De un ejemplo de un problema que solucionaría utilizando metaprogramación.

Parte IV:

Integración partes I, II y III. Ejercicios tipo parcial.

Ejercicio 1 - Integrador.

Modelar con diagramas de clases, objetos y secuencias el ej. 2 i) de la **parte I.**

Ejercicio 2 - Integrador.

Revisite los ejercicios 6, 7 y 8 de la **parte I,** construyendo un nuevo modelo para solucionarlos, pero incluyendo todos conceptos vistos en el curso. Esta vez, incluya en su diseño, un diagrama de clases. ¿Cómo se comparan sus soluciones iniciales, con estas nuevas?

Ejercicio 3 - Integrador.

Un chico puede recorrer una a una las chicas que hay en el lugar que se encuentra (un bar, un boliche o su oficina) e intenta seducirlas.

Se dice que todo chico tiene a un seductor dentro. Hay muchas tácticas para seducir a una chica (mediante trucos de magia, recitando poemas, cantando canciones de Calamaro –que en Ucrania es lo más-, etc...) y todo el tiempo se están descubriendo nuevas. La maniobra de seducción hacia una chica puede fallar o tener éxito. Cuando una técnica tiene éxito, el chico le chifla a Roberto Galán (¡este tipo es único y famosísimo!) el cual ahí mismo crea a la pareja.

Cuando una pareja lo decide le pide a una cigüeña que le traiga un bebé al mundo. Hay varias clases de cigüeñas, cada una encargada de traer bebés de distintas regiones del mundo (por ahora sabemos de cigüeñas francesas y vikingas). A una pareja no le importa qué bebé le traigan mientras le traigan un bebé. Los bebés pueden comer y tener distintos estados de ánimo (contentos, llorando, etc…) aunque todo lo que hacen, lo hacen distinto dependiendo de su país de origen (por ejemplo, los franceses comen de a pequeñas mordidas y dejan la mitad, mientras que los vikingos le entran de una y se comen hasta los huesos). Quien decide qué cigüeña le toca a una pareja es Dios.

Los bebés son monitoreados cuidadosamente por una niñera. Siempre que un bebé cambia su estado de ánimo la niñera se entera. Hay niñeras de todo tipo. Por ejemplo, las responsables, que ante cada cambio de ánimo del bebé tratan de actuar para volverlo a dejar contento. Las empáticas, por su lado, se ponen en el mismo estado de ánimo que el bebé (si el bebé llora, ellas lloran también) y también están las irresponsables, que directamente ignoran al crío.

Se pide cumplir con los siguientes requerimientos extras:

1) Extensibilidad para agregar nuevas tácticas de seducción en un chico sin modificar código existente.
2) Extensibilidad para agregar nuevas clases de cigüeñas sin modificar código existente.
3) Extensibilidad para agregar nuevos tipos de bebes sin modificar código existente.
4) Extensibilidad para agregar nuevos tipos de niñeras sin modificar código existente.
5) Control sobre la cantidad de instancias permitidas de Galán y Dios, y facilidad de acceso a sus métodos por otros.

a) Efectuar un diagrama de clases de la situación descripta.

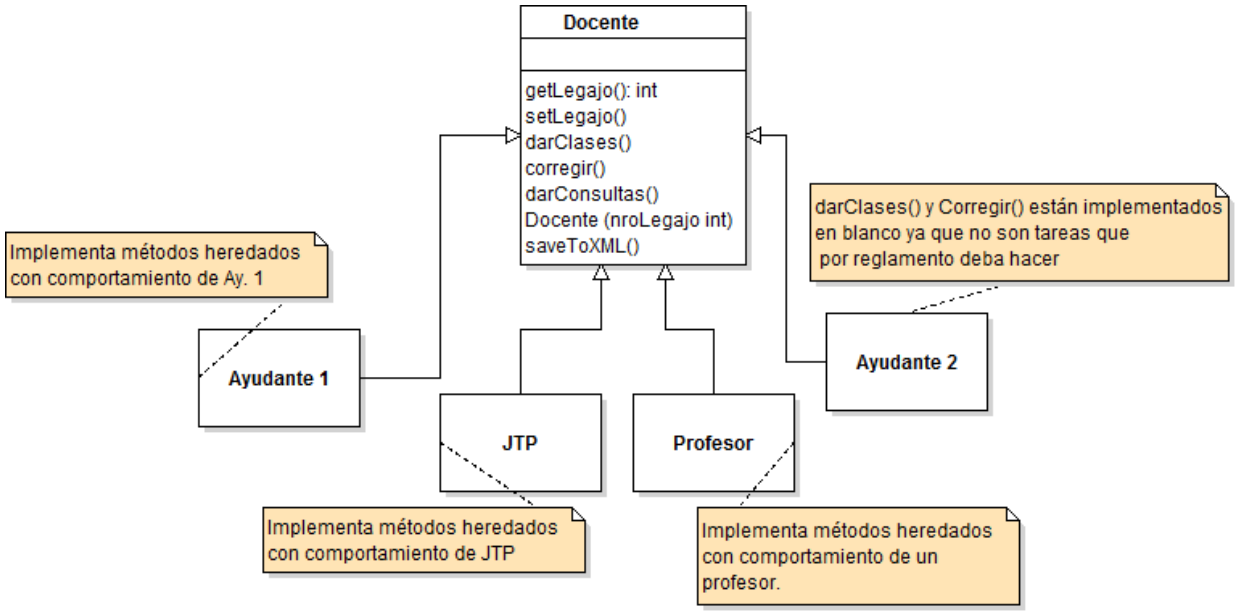
b) i) Efectuar un diagrama de secuencias que muestre el funcionamiento de un chico que desea seducir a chicas en un bar de Ucrania. Hay muchas chicas en el bar, el chico seduce cantando canciones de Calamaro, y recorre a las chicas del mismo hasta encontrar a una en que su táctica sea exitosa (eventualmente siempre lo es ;)) y así formar una pareja…

ii) Efectuar un diagrama de secuencias para mostrar cómo reacciona una niñera responsable cuando un bebe vikingo se larga a llorar.

Ejercicio 4 - Tipo Parcial. Crítica.

Se cuenta con el siguiente diagrama de clases para reflejar el cuerpo educativo de una institución de enseñanza. La institución alienta permanentemente a sus docentes para crecer en su carrera académica porque cree que los ayudantes de hoy serán los profesores del futuro. Además nos informan que los ayudantes de segunda no pueden ni corregir ni dar clases, y sólo están habilitados para dar consultas.

Se cuenta **únicamente** con el diagrama de clases presentado, por lo que en todos los casos puede **realizar las suposiciones que desee** sobre el comportamiento de los mensajes indicándolas claramente en cada caso.



Se pide

i) Indicar posibles errores de diseño clasificándolos en Errores Severos, Medios y Leves.

ii) Proponer una solución posible para todos los errores de severidad grave en no más de tres renglones cada uno.

Ejercicio 5 - Tipo Parcial. Crítica.

El siguiente pseudocódigo modela clientes de una empresa. Un cliente pasa a ser *Preferido* cuando el total de sus facturas supera los $10.000. En ese caso se le devuelve un 25% de las mismas, como premio. Para no discriminar al resto de los clientes, si su facturación supera los $2000, se les devuelve un 10% del total.

```
class Cliente {

  public new(List<Factura> listFac) {
    this.facturas = listFac;
    foreach(f in this.facturas) {
      if (factura.total = null) factura.total = 0;
    }
  }

  public bool addFact(Factura f)
      { this.facturas.add(f); return true; }

  public setFacturas(List<Factura>) { return facturas; }

  public List<Factura>getFacturas() { return facturas; }

  public bool preferido() { return this.preferido == true; }

}

class ClienteRegular < Cliente {

  public new() { this.preferido = false; }

  public float descuentoRegular() { return 0.10; }

  public float totFac() {
    tot = null;
    foreach(f in this.facturas) { tot += f.total; }
    return tot;
  }

}

class ClientePreferido < Cliente {

  public new() { this.preferido = true; }

  public float descuentoPreferido() { return 0.25; }
}
```

Adicionalmente, en algún lado puede encontrarse este bloque de código:

```
public float calcTotalFacturas() {
  total = 0;
  if (cliente.class == ClientePreferido) {
    foreach(f in cliente.facturas)
      { total += f.total; }
    return total * cliente.descuentoPreferido();
  }
  else {
    foreach(f in cliente.facturas)
```

```
      { total += f.total; }
    if (cliente.esRegular() && cliente.totFac() > 2000)
      { total = total * cliente.descuentoRegular(); }
    return total;
  }
}
```

Se pide realizar una crítica del pseudocódigo anterior indicando:

a) Criticidad de cada error: Alta, Media o Baja.
b) Para cada error, una consecuencia negativa en no más de dos líneas.
c) Proponer una solución únicamente para los errores de criticidad Alta.

Ejercicio 6 - Tipo Parcial. Crítica.

El siguiente es un modelo de Cuentas Corrientes de un banco en pseudo-código. El banco sólo otorga la Cuenta Corriente Especial para algunos clientes premium. La misma debe tener un co-titular de forma obligatoria.

```
class CuentaCorriente {
      IdCuenta idCuenta;
      DatosPersonales datosTitularCuenta;
      FondoDinero fondos;
      [...] Otros colaboradores internos [...]

      CuentaCorriente(DatosPersonales unTitular, MontoDinero
      montoInicial, [...] //Otros colaboradores esenciales [...]){
            //Se construye la cuenta corriente...
      }

      DNI DameDNITitularCuenta{
            return this.datosTitularCuenta.DameDNI();
      }

      AlfaNumerico DameNumeroCuenta(){
            return this.idCuenta.DameNumeroCuenta();
      }

      void Extraccion(MontoDinero unMonto){
            this.fondos.Debitar(unMonto);
            //Impuesto del 0.06 % sobre la operación.
            this.fondos.Debitar(
                  new MontoDinero (unMonto.DameTotal()*0.006)
            );
      }

      void Deposito(MontoDinero unMonto){
            this.fondos.Acreditar(unMonto);
            //Impuesto del 0.025 % a la operación.
            this.fondos.Debitar(
                  new MontoDinero (unMonto.DameTotal()*0.0025)
            );
      }

      [...]//Otras operaciones[...]
}
```

```
class CuentaCorrienteEspecial extends CuentaCorriente{
      DatosPersonales datosCoTitularCuenta;

      CuentaCorrienteEspecial(DatosPersonales unTitular,
      DatosPersonales unCoTitular, MontoDinero montoInicial , [...]
      //Otros colaboradores esenciales [...]) {
            super(unTitular, montoInicial, [...]//Otros
            parámetros[...]);
            this.datosCoTitularCuenta = unCoTitular;
      }

      DNI DameDNICoTitularCuenta{
            return this.datosCoTitularCuenta.DameDNI();
      }

      void Deposito(MontoDinero unMonto){
            this.fondos.Acreditar(unMonto);
            //Se bonifica el impuesto al depósito en esta cuenta
            especial y por lo tanto no se cobra.
      }
}
```

Se pide:

Si identifica algún potencial problema con el diseño planteado:
a) Explique el problema en detalle en castellano y **ejemplifíquelo con pseudocódigo.**
b) Resuelva el problema construyendo un **diagrama de clases** de un modelo superador del problema indicado en a).
Si en cambio no identifica ninguno:
a) Justifique en detalle porque el modelo cumple con los conceptos vistos en el curso, y ejemplifique su correcto funcionamiento a través de un **escenario** de su uso y su **diagrama de secuencias** asociado (por ej. construcción de cuenta corriente especial con $500 iniciales, luego se le depositan $200 y por último se le extraen $50).

Ejercicio 7 – Tipo Parcial. Diseño.

Se desea modelar una primera versión de una aplicación móvil de entrenamiento físico para celulares. La misma está pensada para **asistir a una persona mientras corre**.

Para ello se permite definir un **plan de ejercicios** que cuenta de múltiples **fases**, cada fase definida por un tiempo de ejercitación y un ritmo de corrida en km/h. Por ej. un plan puede ser de tres fases, la primera de 10 min a 8km/h (trotando), la segunda de 20 min a 20km/h (corriendo) y la tercera de 5 min a 3km/h (caminando).

La aplicación debe llevar el control de en qué fase se encuentra el corredor (mediante el tiempo transcurrido), y su distancia recorrida hasta el momento. Para obtener la posición actual se posee definida la clase GeoLocalizador, con su mensaje Localizar(): PosicionGPS. Dos objetos PosicionGPS pueden restarse para obtener su distancia en km.

Dependiendo del plan definido, la velocidad calculada del corredor y su distancia ya recorrida, **la aplicación debe notificarle al usuario si se debe aumentar, disminuir o mantener el ritmo** para cumplir con el plan, mostrando la velocidad que debe alcanzar para recorrer la distancia implícita pautada. Este cálculo es sólo dentro de una fase, comenzando de nuevo cuando se inicia una nueva. Si la velocidad calculada está dentro de un +-5% de la velocidad estipulada, se debe utilizar una notificación del tipo "Mantener Ritmo".

Por otro lado, en todo momento se debe poder elegir y cambiar el **perfil de ahorro de energía**. En un principio son tres, **bajo consumo, medio y alto**, pero pueden agregarse nuevos en el futuro. En el perfil bajo, las actualizaciones de posición se hacen cada 1 min y las notificaciones son pitidos predefinidos para bajar (1 pitido), mantener (2 pitidos) o aumentar el ritmo (4 pitidos). En el perfil medio el tiempo de chequeo pasa a 30 segundos y las notificaciones son avisos con imagen y voz (por ej. un grito de "Aumente su velocidad!" y una pantalla roja titilante para una notificación del tipo de "Aumentar Ritmo"). En el perfil de alto consumo, los sponsors del proyecto hicieron su trabajo bien, y proveen de videos cortos de Asafa Powell y Usain Bolt arengando en ritmo de reggae que se corra más rápido, o avisándole que si no baja la velocidad se lesiona antes de llegar a los juegos olímpicos (notificación del tipo "Reducir Ritmo"). Adicionalmente el control de posición en este perfil se hace cada 10 segundos.

i) Efectuar al menos un diagrama de objetos, uno de clases, y todos los diagramas de secuencias (junto a sus escenarios asociados en castellano) que sean necesarios para explicar el funcionamiento completo, de punta a punta, de su modelo. Esto incluye su inicialización, uso, finalización, etc... No deje nada sin especificar, ni explicar. También incluya el pseudocódigo de los métodos más relevantes. **Minimice el uso de ifs.**

ii)¿Existe alguna situación donde no haya podido eliminar el if? En caso afirmativo señale la parte de su modelo donde ocurre, y explique por qué no pudo hacerlo. Contrástelo con algún otro caso, de esta misma solución, donde sí haya podido hacerlo.

Ejercicio 8 - Tipo Parcial. Diseño.

Se desea modelar el funcionamiento de un **"countdown",** o cuenta regresiva. Al mismo se le indica un tiempo inicial en segundos, y este va efectuando una cuenta hacia atrás hasta llegar a cero, momento en el que deja de contar.

Se desea que el countdown pueda verse en varios modos (incluso simultáneamente), para proveer mayor usabilidad. En principio están pensados dos modalidades: una simple donde aparece en pantalla la cantidad de segundos que faltan, y otra que lo muestra en formato (horas / minutos) que cuando termina emite una señal sonora. Se deberían poder habilitar o deshabilitar en cualquier momento, incluso con el countdown corriendo. Posiblemente se agreguen nuevos modos en el futuro cercano.

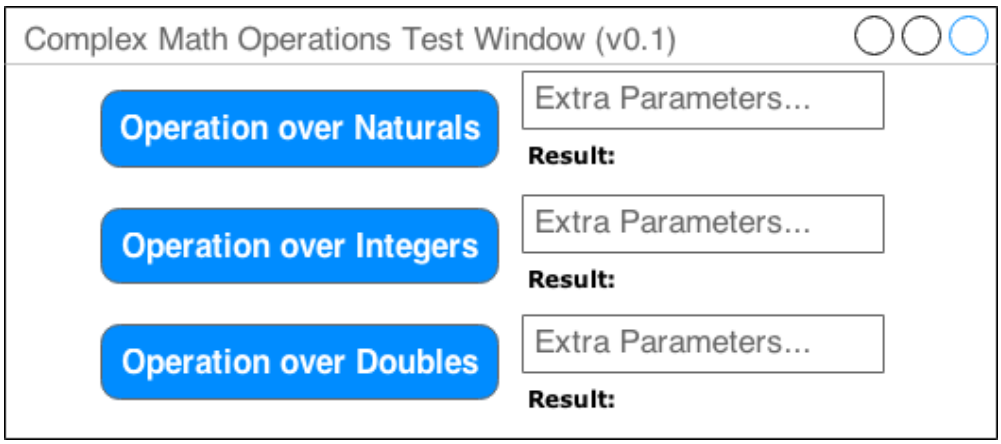
Adicionalmente el countdown debe de poder pausarse y resetear al valor inicial. El reset no funciona sin antes pausar.

Efectuar un diagrama de clases, y todos los diagramas de secuencias (junto a sus escenarios asociados en castellano) que sean necesarios para explicar el funcionamiento completo, de punta a punta, de su modelo. Esto incluye su inicialización, uso, finalización, etc... No deje nada sin especificar. También incluya el pseudocódigo de los métodos más relevantes.

Ejercicio 9 - Tipo Parcial. Diseño.

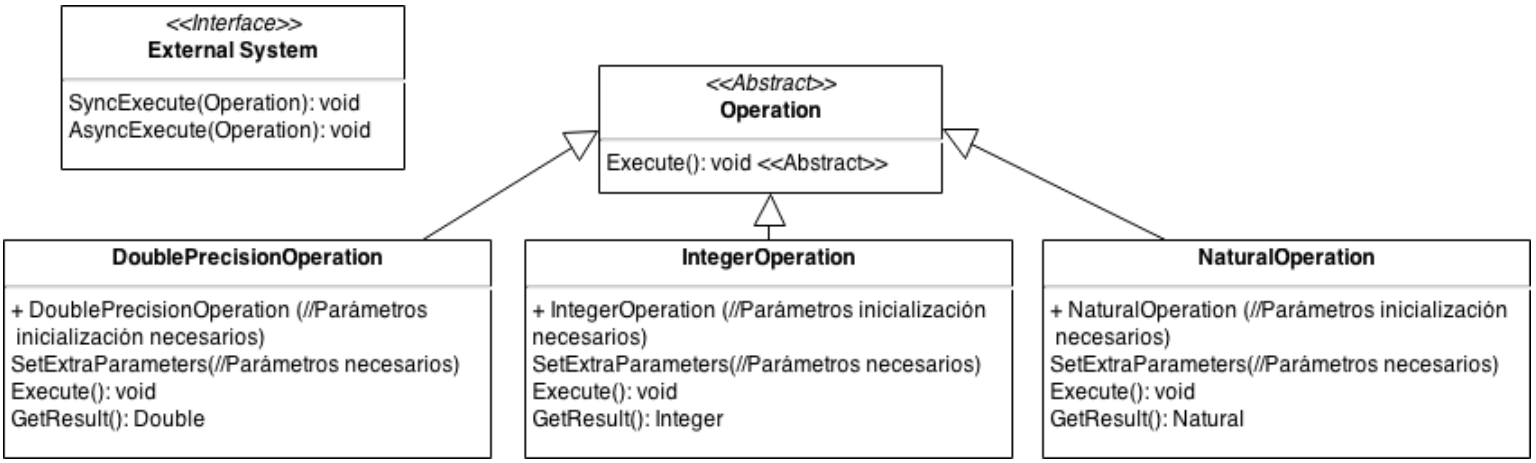
Se está modelando de forma iterativa e incremental una aplicación que requiere de la ejecución de operaciones matemáticas muy complejas y costosas computacionalmente.

En este **primer prototipo** se construyó una interfaz gráfica, por el momento con tres botones, que ejecutan tres operaciones diferentes. La primera utiliza números naturales, la segunda enteros y la tercera números de punto flotante de doble precisión. Cada operación permite ingresar parámetros extra a través de los *textboxes* proporcionados, y muestran el resultado de la operación a través de los *labels* "Result:" correspondientes.

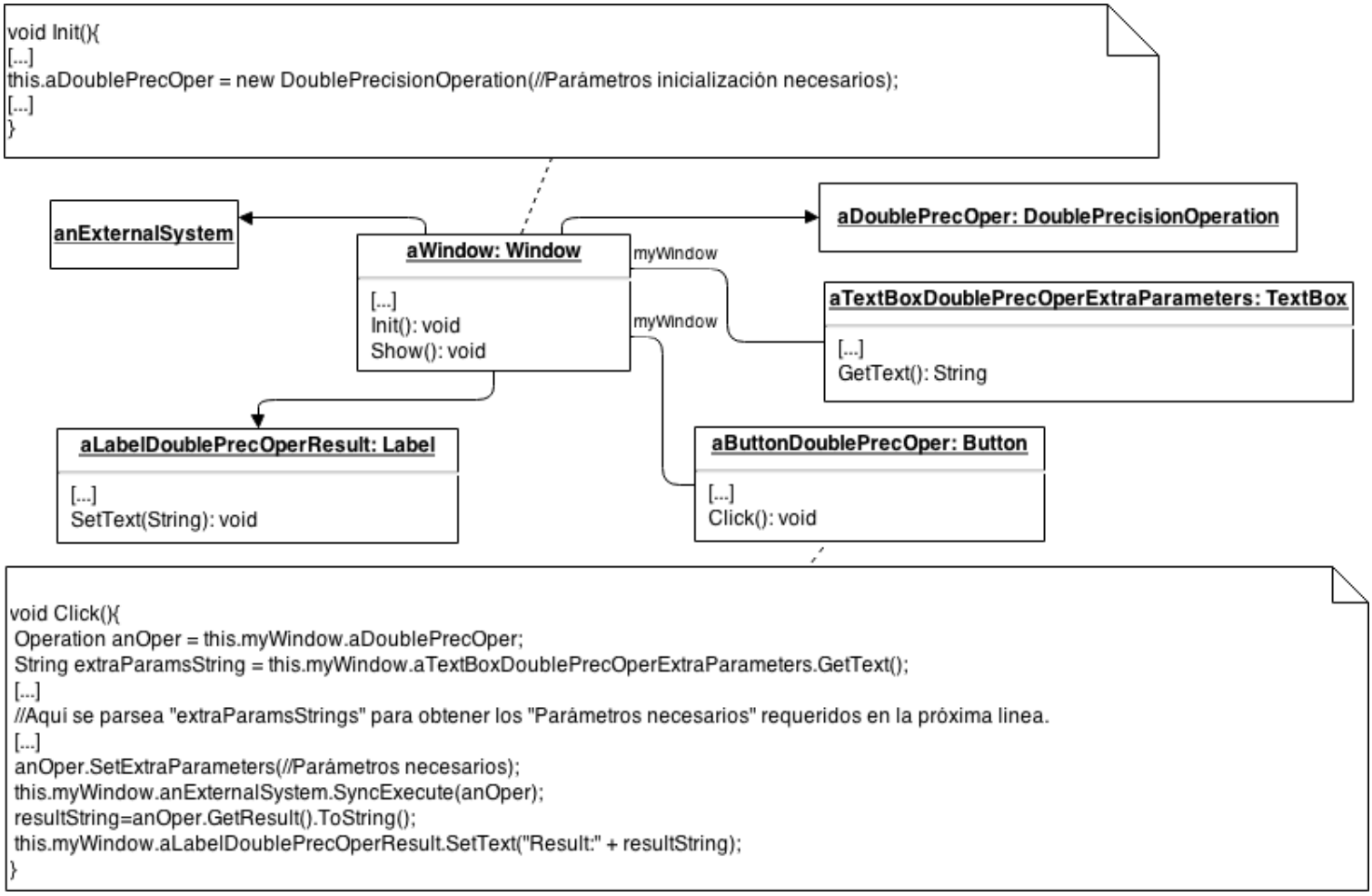


Como las operaciones son muy costosas, y requieren de mucho poder de cálculo, se requiere poder ejecutarlas en servidores externos muy potentes. Para abstraer tal situación se confeccionó la siguiente interfaz "External System", que permite ejecutar operaciones de forma *sincrónica* o *asincrónica* remotamente en tales servidores. Para ello también se definió la clase abstracta "Operation" que representa a las operaciones actuales, y potenciales operaciones futuras.

(Ignore los signos "+" de los diagramas).



Luego alguien diseñó el siguiente diagrama de objetos para el funcionamiento del prototipo actual (sólo se muestra el diseño para la operación de punto flotante de doble precisión. Las otras dos se resuelven de forma equivalente).



Si bien el funcionamiento resultó correcto, lamentablemente en el caso de la operación de punto flotante, la misma tarda tanto tiempo en terminar y arrojar un resultado (orden de los 10 a 15 minutos) que "traba" la interfaz gráfica y por lo tanto la potencial ejecución de las otras dos operaciones.

Se pide:

a) Utilizando los conceptos vistos en la materia, **corregir** el diseño presentado para evitar que ejecutar la operación de punto flotante "trabe" la interfaz, y se pueda potencialmente ejecutar otras operaciones incluso sin haberse reportado su resultado. Para ello utilice todos los **diagramas de objetos, clases, pseudocódigo y explicaciones en castellano** que considere necesarios.

Nota 1: No puede modificar la clase abstracta "Operation", ni la interfaz "External System" para realizar su tarea.
Nota 2: La operación inicial debe mostrar su resultado una vez finalizado.
Nota 3: El AsyncExecute() se encarga de ejecutar el Execute() de la operación de forma asincrónica.

b) Confeccione todos los **diagramas de secuencias/explicaciones en castellano/escenarios** que considere necesarios para modelar todo lo que ocurre desde que se presiona el botón de la operación de punto flotante hasta que el resultado de la misma se muestra en el *label* resultado correspondiente, en el nuevo modelo.

Ejercicio 10 - Tipo Parcial. Diseño.

Desde la Federación Argentina de Automovilismo nos han encargado el desarrollo de un nuevo simulador de coches de carrera.

El simulador podrá responder a las clásicas operaciones de aceleración, frenado, girar a izquierda y girar a derecha.

Para facilitar la interacción con los usuarios se ha dispuesto su funcionamiento bajo 3 modalidades: principiante, intermedio, y profesional, en niveles crecientes de dificultad. En otras palabras, frenar y girar a izquierda en una curva cerrada será más sencillo en el nivel principiante que en el nivel profesional. Será posible cambiar la modalidad en cualquier momento de la simulación. Asimismo, nos comentan que están evaluando incluir nuevos modos de funcionamiento en el futuro. Por ejemplo, la modalidad Fangio-Intense buscará desafiar al máximo las habilidades del piloto.

Existe otro factor que influye en los movimientos del auto además de las distintas modalidades, que son las condiciones climáticas. Por el momento se están considerando simulaciones bajo lluvia, bajo granizo, bajo extremo calor o normales. Luego, será diferente girar a izquierda en cada condición climática. Similarmente, lo mismo sucederá en cada movimiento del auto. Nuevamente, será posible cambiar la situación climática en todo momento.

Resumiendo, cada movimiento del auto será determinado tanto por la modalidad del simulador (principiante, intermedio, y profesional), como por la condición climática (normal, calor, lluvia, granizo). Por omisión, el simulador comenzará en modalidad principiante, y en condiciones climáticas normales.

Finalmente, la simulación deberá constatar la cantidad de veces que el piloto se fue de pista, el tiempo promedio, y el tiempo de la mejor vuelta.

Se pide: **(Nota: no interesan los detalles algorítmicos de cómo son procesados los movimientos del simulador)**

i) Efectuar un diagrama de clases que modele la situación planteada.
ii) Efectuar un diagrama de secuencias y otro de objetos de la siguiente situación: un usuario gira a la izquierda con el simulador en modalidad experta bajo extremo calor.
iii) Suponga ahora que se agrega un nuevo tipo de movimiento del auto: estacionamiento. Describa cómo impacta este nuevo requerimiento en el modelo presentado en los puntos i y ii.

Ejercicio 11 - Tipo Parcial. Diseño.

La empresa Artistic Delivery ha desarrollado un proyecto para construir drones inteligentes para realizar sus entregas. En particular, se desea por el momento enfocarse en el componente de comunicación.

Como la principal responsable de la empresa, la dra Marina Abramovic, es una amante de la naturaleza, desea que dicho componente minimice el impacto ambiental. En tal sentido la funcionalidad para enviar y recibir mensajes ha sido concebida para funcionar bajo dos modalidades. Cuando el tiempo lo permite, estas tareas se realizan de una manera eco-friendly, ahorrando recursos y minimizando así la contaminación ambiental. En cuanto se nubla o no se cuenta con la luz suficiente, la comunicación funcionará de la manera tradicional, sin tener en cuenta los efectos contra el medio ambiente.

Cada cierto período de tiempo (inicialmente 10 segundos, pero puede cambiarse según se requiera) el dispositivo obtiene la información del estado climático y en base a dicha información irá actualizando su modalidad de transmisión.

Para obtener la información climática el dispositivo se comunica con un satélite geostacionario cuyas coordenadas son enviadas al momento de iniciarse el dispositivo. Sin esta información el drone no puede funcionar. Las coordenadas del satélite pueden cambiarse libremente en cualquier momento.

Dentro del tipo de mensajes que se pueden enviar a un dron, la dra. Abramovic solicita que nos enfoquemos en los comandos que puede recibir uno de estos bonitos artefactos. Se puede tratar tanto de mensajes simples (moverse, frenar, girar, elevar) o secuenciales, que forman una sucesión de comandos que debe procesarse de manera atómica, pudiendo éstos ser a su vez, simples o secuenciales. Por ejemplo, el mensaje M1= **{Girar}** es un mensaje simple, el mensaje M2 = **{Elevar, Girar, Frenar}** es uno secuencial y el mensaje M3 = **{Girar, {Elevar, Girar, Frenar}}** es un mensaje secuencial que está construido con el mensaje simple M1, y el mensaje secuencial M2.

Si bien existe mucha funcionalidad dentro de los mensajes, nos piden que nos concentremos únicamente en el tamaño de los mismos. Nos informan además, que los mensajes simples ocupan una unidad de tamaño (UT), mientras que los secuenciales se compactan y terminan ocupando la mitad del tamaño de los mensajes que lo componen. Volviendo al ejemplo mencionado, el mensaje M1 **{Girar}** tiene tamaño 1 UT, el mensaje M2 **{Elevar, Girar, Frenar}** 1,5 UT, mientras que el mensaje M3 **{Girar, {Elevar, Girar, Frenar}}** tiene entonces 1,25 UT. Esto se corresponde a la suma de los mensajes que lo componen (M1: 1 UT y M2: 1,5 UT) dividido 2.

Nota: la implementación concreta de las funcionalidades de enviar y recibir en ambas modalidades no es relevante para el presente ejercicio. Lo mismo para la funcionalidad de decidir el modo en base a la información climática recibida.

1) Se pide diseñar un modelo que cubra lo requerido a través de los siguientes puntos

a) Realice un diagrama de secuencia y uno de objetos para el siguiente escenario: el dispositivo de comunicación se encuentra funcionando en modo

tradicional, se obtiene información del estado climático y se cambia a modo ecológico. Luego, se pide enviar un mensaje y se transmite en la modalidad esperada.

b) Realice un diagrama de secuencias y uno de objetos para el siguiente escenario: se pide el tamaño al mensaje **{Girar, {Elevar, Girar, Frenar}}** y se obtiene como resultado 1,25 UT.

c) Realice un diagrama de clases que cumpla con lo detallado en el enunciado, y respete lo realizado en los puntos a) y b). Incluir todos los mensajes necesarios, incluyendo aquellos que crean instancias.

d) Justifique y explique su modelo presentado en los puntos a), b), y c).

2) Suponga ahora nos comunican que se agrega una nueva modalidad, para realizar la comunicación con el cielo parcialmente nublado. Detalle cómo modificaría lo modelado en el punto 1 para adaptarse a este nuevo requerimiento.

Ejercicio 12 - Tipo Parcial. Diseño.

Una importante cadena de cines desea desarrollar un sistema para incorporar una máquina expendedora de entradas que agilice el proceso de venta. Dentro de toda la funcionalidad de la máquina nos piden concentrarnos por el momento en el sistema encargado procesar el cobro de las entradas.

Se sabe que el sistema operará únicamente con tarjeta de crédito. Por el momento el sistema trabajará con las tarjetas "La alegría de consumir" y "Viva la vida", aunque se espera sumar nuevas tarjetas a corto plazo. Además se sabe que funcionará bajo dos modalidades: generoso y tradicional. En la modalidad tradicional el costo de la entrada para cualquier película será de $100, que es el valor que ha establecido la cadena. En el modo generoso se aplicará un descuento según el día de la semana y la tarjeta de crédito utilizada, como se detallará más adelante. Nos informan que desde los otros sistemas que forman la máquina se podrá cambiar la modalidad al iniciarse una transacción según alguna regla que establezca la cadena. Si bien estos otros sistemas están fuera de alcance, el sistema de cobro a desarrollar debe permitir cambiar de una modalidad a otra. La opción por defecto es funcionar en modo tradicional.

Para la modalidad generoso se cuenta con la siguiente información. La tarjeta "Viva la vida" ofrece un 10% de descuento los lunes, martes y viernes, y 18% los días restantes. En cambio, la tarjeta "La alegría de consumir" ofrece 2% los sábados y domingos, y 3% los días de semana.

Por ejemplo, Francisco vio el martes pasado la película "Calurosa Luna de Miel en Brasil" protagonizada por Nicolas Cage, y pagó con la tarjeta "Viva la Vida". Tuvo suerte ya que el sistema de cobro estaba funcionando en modalidad generoso. Obtuvo entonces un 10% de descuento, por lo que pagó $90. En cambio, Nicolás fue a ver el documental "Manu y los Spurs, maridaje perfecto". Nicolás no tuvo tanta suerte ya que el sistema de cobros funcionaba en la modalidad tradicional, por lo que pagó los $100.

Para efectuar el cobro sobre la tarjeta de crédito, se enviará el monto y la tarjeta al sistema Comunicación Bancaria, quien llevará adelante la operación. Si bien este

sistema está fuera de alcance se conoce que para efectuar el cobro se deberá invocar a ProcesarCobro(monto, tarjeta).

Se pide diseñar un modelo que cubra lo requerido a través de los siguientes puntos:

a) Realice un diagrama de secuencia y uno de objetos para el escenario donde Francisco fue al cine con el sistema de cobro funcionando en modalidad generoso y pagó $90 su entrada.

b) Realice un diagrama de clases que cumpla con lo detallado en el enunciado, y respete lo realizado en el punto anterior. Incluir todos los mensajes necesarios.

c) Justifique y explique su modelo presentado en los puntos a) y b)

Suponga ahora que la tarjeta "Compro, luego existo" se incorpora a la cadena. Describa brevemente cómo modificaría el modelo presentado en el punto 1) para adaptarse a este nuevo requerimiento.

Ejercicio 13 - Tipo Parcial. Diseño.

¡Felicitaciones! Después de varias entrevistas, la empresa ACME finalmente los contrató para implementar su sistema de ventas de productos on-line. Esto es lo que les comentaron en la primera reunión:

> *Nos dedicamos a vender productos, que pueden ser tanto insumos como servicios. Los servicios tienen una duración especificada.*
> *Después de un análisis de marketing nos dimos cuenta que a nuestros clientes les encantan los productos organizados por categorías. Si esto les gusta, entonces segurísimo que las ventas se incrementarían mucho si las categorías, además de contener productos, pueden contener otras categorías adentro: ¡más es mejor! ¡Es muy importante que esto esté implementado! La idea es que nuestros clientes ingresen al sitio web, naveguen por las categorías y lleguen al producto que les interese comprar.*
> *¡Ah! Por ahora, alcanza solamente con mostrar el nombre y el precio, teniendo en cuenta que los servicios también deben mostrar su duración.*

Un ejemplo de lo que se debería ver en el sitio es:

```
Productos baratos
Productos caros
      Insumo 1 $500
      Servicio 1 $600 (30 días)
      Productos muy caros
            Insumo 2 $1500
```

Se pide:

a) Realizar un diagrama de secuencias (colaboraciones) donde se genere el listado de productos que se muestra en el ejemplo, sin importar el formato.
b) Representar el ejemplo en un diagrama de objetos.
c) Como era de esperarse el sistema fue todo un éxito. Tanto, que algunos insumos quedaron fuera de stock. Se decidió que no tengan precio hasta reponerlos, así que en lugar del precio se debe mostrar el mensaje

"disponible próximamente". Sin modelar ni implementar nada, comentar cómo implementarían este cambio de modo que el diseño se vea afectado lo menos posible.

Notar que:

- La lista anterior es un ejemplo nada más. No significa que tenga que existir una categoría "Productos baratos" ni "Productos caros". Esto lo va a configurar el cliente directamente desde el sistema on-line (fuera del alcance del ejercicio)
- No hace falta que los elementos tengan sangría ("indentación").
- Pueden haber categorías vacías ("Productos baratos")
- Pueden haber categorías dentro de categorías ("Productos muy caros")
- Los servicios tienen información adicional para mostrar: su duración.

Apéndice A:

Preguntas teóricas.

Responda Verdadero o Falso. Justifique cada caso.

a) La mejor forma de reutilizar código es utilizando herencia.
b) Todos los objetos deben ser completamente inmutables. Es decir, no deben cambiar desde su creación. Brinde un ejemplo.
c) Desde el punto de vista del paradigma de objetos a la relación de clasificación (herencia) es mejor verla como "se comporta cómo" en vez de "es un".
d) Un buen diseño está basado exclusivamente en el uso de patrones.
e) Un objeto debe recibir todos sus colaboradores internos al momento de su creación.
f) El diagrama de clases es el mejor para reflejar el paso del tiempo y la evolución de los objetos.
g) El mejor mensaje de creación de instancia es aquel que no recibe parámetros.
h) Un buen diseño no debe proveer métodos que cambien la esencia de un objeto.
i) Los patrones de diseño son ESTRUCTURAS de objetos que resuelven problemas habituales.
j) Deben evitarse los métodos que devuelven NULL.
k) En los lenguajes de clasificación no es posible usar el patrón Decorator si no se cuenta de antemano con un padre abstracto o interfaz de la clase que quiero decorar.
l) Es conveniente tener objetos que no sean válidos porque así es más fácil representar situaciones no contempladas.
ll) Todos los objetos deben estar completos desde el momento de su creación.
m) La reutilización de código es el principal motivo para utilizar subclasificación.
n) Según el paradigma de objetos los programas son secuencias de mensajes.
ñ) El patrón Decorator sirve para implementar inicialización Lazy.
o) Una buena medida para saber si un diseño es mejor que otro es contar la cantidad de patrones que utiliza cada uno y elegir el que utilice la mayor cantidad.
p) el patrón Abstract Factory se utiliza para crear instancias de clases abstractas.

Desarrollar:

a) Describa tres beneficios que brinda el polimorfismo en un paradigma de objetos.
b) Explique dos ventajas con ejemplos concretos de no más de tres renglones de usar patrones de diseño (las ventajas tienen que ver con usar patrones al diseñar y no con el uso de uno de estos en particular).

c) ¿Qué es la sub-clasificación? De ejemplos de un buen y un mal uso de la misma.
d) ¿Qué es el "intent" o intención de un patrón? ¿Cómo se diferencia de la estructura de un patrón?
e) ¿Considera una buena política mecanismos como el "final" de Java, que establece que una determinada clase no pueda ser heredada? ¿Por qué?
f) Explique en sus propias palabras la relación entre esencia e inmutabilidad de los objetos. Dé ejemplos.
g) Explique brevemente, y en sus propias palabras qué significa que un objeto conozca a otro. Brinde ejemplos.
h) El cambio es una característica esencial del software.
i) ¿Qué es el código repetido? Brinde un ejemplo y muestre cómo solucionarlo.
j) ¿Qué ente del mundo real representa una clase? ¿Cuál es su principal responsabilidad?
k) ¿Qué es el polimorfimo y para qué se utiliza? Brinde un ejemplo.
l) ¿Qué define el comportamiento de un objeto?
ll) ¿Por qué un objeto debe ser válido desde el momento de su creación? ¿Qué ventajas y desventajas acarrea esta decisión?
m) Explique en lenguaje natural los conceptos de Cohesión y Acoplamiento en no más de diez líneas en total.
n) ¿Qué tipo de conocimiento busca transmitirse en un diagrama de clases? ¿Y en uno de objetos? Muestre un ejemplo de una situación que no puede reflejarse en cada uno de los dos diagramas.
ñ) ¿Qué ente de la realidad está representando una clase abstracta?
o) Explique en sus propias palabras la diferencia entre usar herencia e interfaces para compartir conocimiento. Dé ejemplos, ventajas y desventajas.