PRACTICO N° 5

EJERCICIO 1. En un videojuego se modelan criaturas que tienen la capacidad de jugar o descansar. En el momento que se crea una criatura a su energía se asigna el valor del atributo de clase. Cuando una criatura juega consume 10 unidades de energía, cuando descansa recupera 100. Los servicios **igual** y **mayor** comparan las criaturas por nombre, utilizando el método compareTo provisto por la clase String. Implemente el siguiente diagrama de acuerdo a la especificación.

Criatura
<<atributos de clase>> energiaMaxima=1000 **<<atributos de instancia>>** nombre:String energía: entero
<<constructor>> Criatura (nom:String) **<<Comandos>>** jugar() descansar() **<<Consultas>>** obtenerNombre(): String obtenerEnergia(): entero igual(c:Criatura) : boolean mayor(c:Criatura): boolean

EJERCICIO 2. Una organización dedicada a la preservación del medio ambiente mantiene información referida a cada especie animal de la que se realiza seguimiento. Cada especie está caracterizada por su nombre científico, la cantidad de hembras y la cantidad de machos que se registran en la actualidad y el ritmo de crecimiento anual de la población de acuerdo a lo ocurrido en los últimos 10 años. La cantidad de hembras y machos tiene que ser siempre un valor no negativo, el ritmo puede ser cualquier valor real.

Los tres valores numéricos se inicializan en 0 y luego pueden establecerse o actualizarse con valor dado. Los comandos establecerMachos y establecerHembras controlan que el parámetro sea positivo, en caso contrario no modifican el valor del atributo. Los comandos actualizarMachos y actualizarHembras suman el valor del parámetro al atributo de instancia. Pueden recibir un parámetro negativo, pero el valor del atributo nunca va a ser menor a 0.

La consulta clone retorna una especie con el mismo estado interno que la especie que recibe el mensaje. Observe que luego de crear una especie debe establecer los valores de modo que el estado interno de la especie que retorna sea el mismo que el de la especie que recibe el mensaje.

La organización está interesada en calcular para una especie particular: la población total, el nivel de riesgo de extinción (verde si el ritmo de crecimiento es positivo, amarillo si es nulo y rojo si es negativo), la población estimada para dentro de una cantidad dada de años, cuántos años serán necesarios estimativamente para que la población alcance un valor dado. Se desea determinar también si en una especie dada es mayor el número de hembras o de machos y entre dos especies diferentes cuál tiene mayor ritmo de crecimiento.

a) A partir del diagrama y la especificación **implemente** la clase *Especie* en Java encapsulando atributos y comportamiento para modelar la situación planteada. Incluya todos los métodos auxiliares que considere oportunos, como así también los métodos triviales que no estén incluidos en el diagrama.

b) Escriba una clase tester que verifique los servicios provistos por Especie con valores ingresados por el usuario para nombre y ritmo y generados al azar para machos y hembras dentro de un rango establecido.

Especie
<<Atributos de instancia>> nombre: String machos: entero hembras: entero ritmo: real
<<Constructor>> Especie(nom:String) <<Comandos>> establecerHembras(h:entero) establecerMachos(m:entero) establecerRitmo(r:real) actualizarHembras(h:entero) actualizarMachos(m:entero) actualizarRitmo(r:real) <<Consultas>> poblacionActual(): entero poblacionEstimada(anios:entero): entero anios(pob:entero): entero riesgo(): String masHembras(): boolean mayorRitmo(est: Especie): Especie clone():Especie toString(): String

EJERCICIO 3. En una Biblioteca el préstamo de un libro se modela de acuerdo al siguiente diagrama (se utiliza la clase Fecha del práctico 4[1], con una consulta adicional: la consulta equals):

Prestamo	**Libro**
<<Atributos de instancia>> libro:Libro socio:String fechaPrestamo:Fecha dias:entero fechaDevolucion:Fecha	**<<Atributos de instancia>>** nombre:String autor:String editorial:String categoria:char
<<Constructor>> Prestamo(l:Libro, fp:Fecha, d:entero, s:String) **<<Comandos>>** establecerFechaDevolucion (fd:Fecha) **<<Consultas>>** obtenerLibro(): Libro obtenerFechaPrestamo():Fecha obtenerFechaDevolucion():Fecha estaAtrasado(f:Fecha):boolean	**<<Constructor>>** Libro(n,a,e:String, c:char)

1 En caso de no tener la clase implementada del práctico 4 se puede descargar una versión completamente implementada de la página web de la materia en el apartado “Material Adicional”.

penalizacion():Fecha	

La clase Libro brinda los comandos y consultas triviales.
En la clase Prestamo:

- establecerFechaDevolucion recibe como parámetro la fecha en la que efectivamente se realizó la devolución del libro.
- obtenerFechaDevolución retorna la fecha en la que efectivamente se realizó la devolución del libro.
- estaAtrasado recibe como parámetro la fecha actual y retorna verdadero si el libro no se devolvió y ya debería haberse devuelto de acuerdo a la fecha de préstamo y la cantidad de días.
- penalizacion calcula la cantidad de dias de penalización y devuelve la fecha hasta la que corresponde aplicar la penalización, a partir de la fecha de devolución, que tiene que estar ligada. La penalización es de 3 días si se atrasó menos de una semana, 5 días si se atrasó menos de tres semanas y 10 días si se atrasó 3 semanas o más. Si el libro tiene categoría A los días de penalización se duplican.

a. Implemente el diagrama complete.
b. Implemente una clase tester con algunos valores fijos y que le permita al usuario ingresar otros de prueba.

EJERCICIO 4. En un hospital se mantiene información referida a las cirugías de acuerdo al siguiente diagrama de clases:

Paciente
<<atributos de instancia>> nombre : String obraSocial : String sexo : char fechaNac : Fecha
<<consultas>> igual (pac : Paciente) : boolean edad (act :Fecha): entero

Cirugía
<<atributos de instancia>> fecha : Fecha minutosDuracion : entero quirofano : entero pac : Paciente
<<consultas>> igual(c:Cirugia):boolean igualOS(c:Cirugia):boolean

igual (pac : Paciente) : boolean retorna true sí y solo sí el estado interno del paciente que recibe el mensaje y el estado interno del paciente que pasa como parámetro son equivalentes.
igual(c:Cirugia):boolean retorna true si la cirugía que recibe el mensaje y la que pasa como parámetro tienen los mismos valores en minutosDuración, quirófano y pac y corresponden a fechas equivalentes. Es decir los atributos pac tiene la misma identidad en las dos cirugías.
igualOS(c:Cirugia):boolean retorna true si la cirugía que recibe el mensaje y la que pasa como parámetro se practican a pacientes que tienen valores equivalentes en el atributo obraSocial.

a. Implemente las clases propuestas en el diagrama incluyendo los servicios triviales de cada clase.
b. Diseñe casos de prueba adecuados para verificar los servicios.

EJERCICIO 5. En un campeonato de fútbol por cada partido ganado se obtienen 3 puntos y por cada empate se logra 1. Cada equipo tiene un nombre, un capitán, una cantidad de partidos ganados, otra de empatados y otra de perdidos, una cantidad de goles a favor y otra de goles en contra. El capitán es un jugador que tiene un nombre, un número de camiseta, un número que representa la posición en la que juega, la cantidad de partidos jugados y la cantidad de goles convertidos en el campeonato.
Para un jugador se desea calcular el promedio de goles por partido y dado otro jugador, cuál es el que hizo más goles. Para un equipo se desea calcular los partidos jugados y los puntos obtenidos. Además para otro

equipo dado es necesario decidir cuál es el equipo con mayor puntaje y cuál es el capitán con más goles. Si dos equipos tienen los mismos puntos, se devuelve el que tiene mayor cantidad de goles a favor y si también hay coincidencia, el que tiene menos goles en contra. Si hay coincidencia se devuelve uno cualquiera.

a) A partir del diagrama de clases y la especificación implemente dos clases en Java encapsulando atributos y comportamiento. Incluya dentro del comportamiento los comandos para modificar y consultar cada atributo.

Jugador
<<Atributos de instancia>> nombre: String nroCamiseta: entero posicion: entero golesConvertidos: entero partidosJugados: entero
<<Constructor>> Jugador(nom:String) **<<Comandos>>** aumentarGoles(n:entero) aumentarUnPartido() **<<Consultas>>** promedioGolesXPart(): entero jugConMasGoles(j: Jugador): Jugador masGoles(j:Jugador): boolean toString(): String

masGoles(j:Jugador): boolean
devuelve true si el jugador que recibe el mensaje tiene más goles que el jugador que corresponde al parámetro.

Equipo
<<Atributos de instancia>> nombre: String capitan: Jugador pG,pE,pP: entero gFavor, gContra: entero
<<Constructor>> Equipo(nom:String, cap: Jugador) **<<Comandos>>** incrementarPG(jugoElCap: boolean) incrementarPE(jugoElCap: boolean) incrementarPP(jugoElCap:boolean) aumentarGfavor(total, delCap: entero) aumentarGContra(total: entero) **<<Consultas>>** partidos(): entero puntos(): entero mejorPuntaje(e: Equipos): Equipo capitanConMasGoles(e: Equipo): Jugador

incrementarPG(jugoElCap: boolean)
incrementarPE(jugoElCap: boolean)
incrementarPP(jugoElCap:boolean)
Aumentan en 1 los partidos del equipo y si corresponde envía un mensaje al capitán para que incremente en 1 sus partidos.

aumentarGfavor(total, delCap: entero)
Aumenta los goles del equipo y si corresponde envía un mensaje al capitán para actualizar sus goles.

b) Considerando el mismo modelo para jugador implemente una nueva clase Equipo para la siguiente especificación:
   - ***incrementarPG() incrementarPE() incrementarPP():*** *Aumenta en 1 los partidos del equipo.*
   - ***incrementarGFavor(total: entero):*** *Aumenta los goles del equipo.*

Equipo
<<Atributos de instancia>> nombre: String capitan: Jugador pG,pE,pP: entero gFavor, gContra: entero
<<Constructor>> Equipo(nom:String, cap: Jugador) **<<Comandos>>** incrementarPG() incrementarPE() incrementarPP() aumentarGfavor(total: entero) aumentarGContra(total: entero) **<<Consultas>>** partidos(): entero puntos(): entero mejorPuntaje(e: Equipos): Equipo capitanConMasGoles(e: Equipo): Jugador

incrementarPG()
incrementarPE()
incrementarPP()
Si jugó el capitán requiere que la clase cliente incremente el número de partidos jugados por el capitán.

aumentarGfavor(total: entero)
Si el gol fue del capitán requiere que la clase cliente aumente el número de goles del capitán.

c) Establezca casos de prueba adecuados e implemente una clase para testear cada alternativa utilizando dichos valores.

Objetivos del práctico. Clases asociadas. La clase String. alternativas de Diseño. Tester con casos de prueba establecidos en el diseño. Tester con valores generados al azar. Tester con valores ingresados por el usuario.