PRACTICO N° 2: Objetos y Clases

Ejercicio 1: Analice si las siguientes definiciones son correctas

I. Desde el punto de vista del diseño de un sistema orientado a objetos, una clase es un patrón que establece los atributos y el comportamiento de un objeto.

II. En la implementación de un sistema orientado a objetos, una clase es un módulo de software que puede desarrollarse con cierta independencia del resto de los módulos.

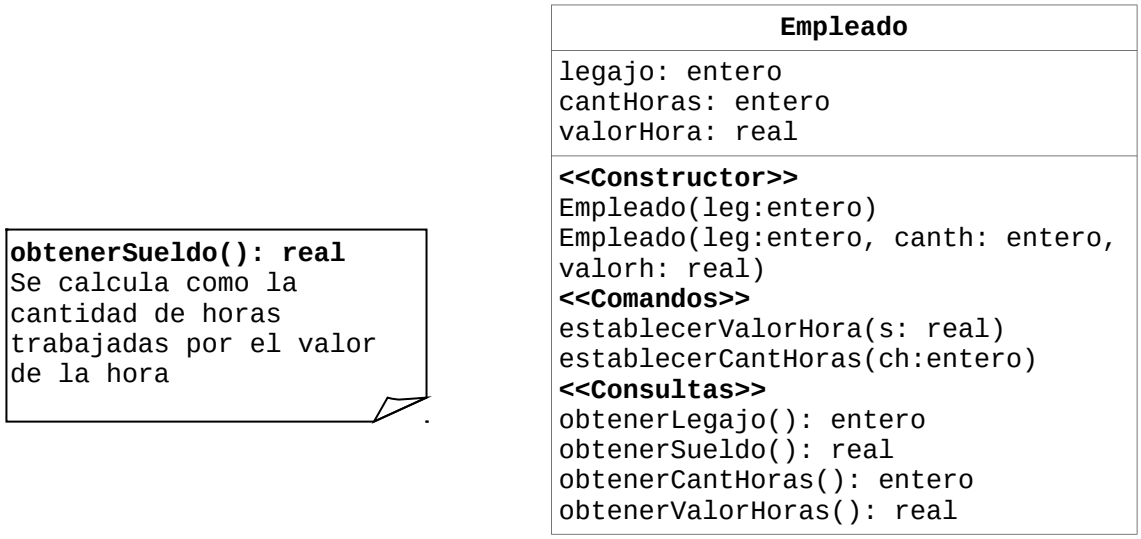
III. Un constructor es un método que se invoca cuando se crea un objeto.

IV. Un comando es un método que no retorna un resultado.

V. Una consulta es un método que no modifica los valores del objeto.

Ejercicio 2: Una empresa desea llevar el registro de sus empleados para poder realizar la liquidación de sueldos.

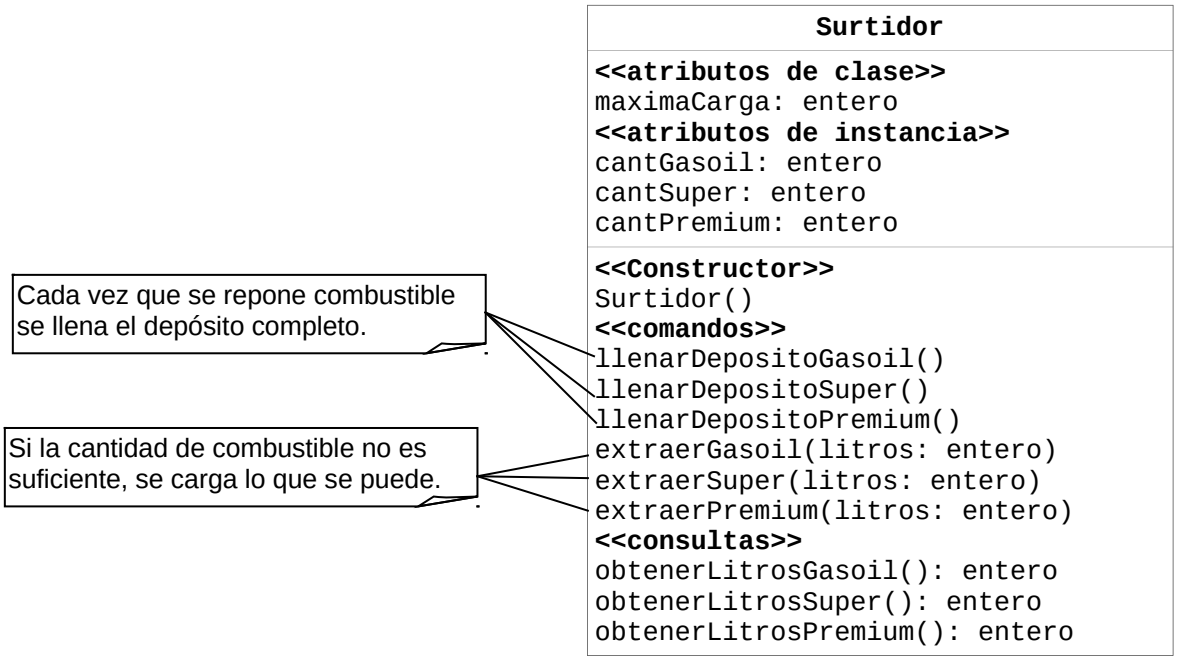
a) Implemente en Java la clase Empleado modelada por el siguiente diagrama:



b) Escriba una clase Tester con un método main() que solicite al usuario los datos de un empleado (legajo, cantidad de horas trabajadas y valor de la hora), cree un objeto de la clase Empleado usando el constructor con tres parámetros y a continuación muestre por pantalla el Legajo y el sueldo del empleado.

c) Escriba una clase Tester con un método main() que solicite al usuario los datos de un empleado (legajo, cantidad de horas trabajadas y valor de la hora), cree un objeto de la clase Empleado usando el constructor con un parámetro, modifique la cantidad de horas y el valor de la hora usando los métodos establecer y a continuación muestre por pantalla el Legajo y el sueldo del empleado.

Ejercicio 3: Una estación de servicio cuenta con surtidores de combustible capaces de proveer Gasoil, Nafta Super y Nafta Premium 2000. Todos los surtidores tienen capacidad para almacenar un máximo de 20000 litros de cada combustible. En cada surtidor se mantiene registro de la cantidad de litros disponibles en depósito de cada tipo de combustible, esta cantidad se inicializa en el momento de crearse un surtidor con la cantidad máxima. En cada surtidor es posible cargar o reponer combustible. En ocasiones la cantidad de un tipo de combustible particular en un surtidor específico puede no ser suficiente para completar una carga, en ese caso se carga lo que se puede y cuando el depósito del surtidor queda vacío si no se completó la carga se muestra un mensaje. Cuando se repone un combustible en el surtidor, se llena el depósito completo de ese combustible. Cada surtidor puede modelarse con el siguiente diagrama:



a) Implemente en Java la clase Surtidor
b) Escriba una clase SimulacionSurtidor con un método main() que permita verificar los servicios provistos por la clase Surtidor de acuerdo al siguiente algoritmo:

```
Algoritmo simulador
n = leer cantidad de iteraciones
repetir n veces testSurtidor

Algoritmo testSurtidor
  mostrar la cantidad actual en el depósito de cada combustible
  opción= número al azar entre 0 y 32
  según opción sea:
    Entre 0 y 9: leer litros a cargar y cargar Gasoil
    Entre 10 y 19: leer litros a cargar y cargar Super
    Entre 20 y 29: leer litros a cargar y cargar Premium
    30: llenar Deposito Gasoil
    31: llenar Deposito Super
    32: llenar Deposito Premium
```

Ejercicio 4. Se conoce como Langostas Mutantes a una variedad de las langostas que viven cerca de las centrales nucleares. Este tipo de langosta tiene dos características que la diferencia del resto de la especie: Por un lado, como todo animal que vive cerca de centrales nucleares, tiene tres ojos; y además su ciclo de reproducción es realmente extraño. La reproducción de la Langosta Mutante se puede describir de la siguiente manera:

Reproducción: La hembra langosta tiene una cría. Sí la cría es hembra entonces la langosta madre finaliza su ciclo de reproducción actual. Sí la cría es macho entonces la langosta automáticamente se reproducirá una vez más siguiendo el comportamiento explicado.
El comando *reproduccionEnLaColonia()* simula la reproducción de cada hembra en la colonia (recuerde que al reproducirse se cambia la cantidad de machos y hembras en la colonia) invocando al método recursivo *reproduccion()*. Tenga en cuenta que las hembras recién nacidas necesitan cierto proceso de maduración antes de reproducirse, es por eso que no pueden reproducirse hasta la próxima invocación del método *reproduccionEnLaColonia()*.
Al crear la colonia hay un solo macho y una sola hembra.

ColoniaLangostasMutantes
<<Atributos de instancia>> machos,hembras:entero
<<Constructor>> ColoniaLangostasMutantes() **<<Comandos>>** reproduccionEnLaColonia() reproduccion() **<<Consultas>>** tenerCria():char tamanioColonia():entero toString():String

reproducción() computa en forma recursiva la reproducción de una hembra.

reproduccionColonia() modifica el número de machos y hembras a partir de la reproducción de cada hembra de la colonia

tenerCria() retorna, en forma aleatoria, un carácter indicando el género de la cría

ejemplaresColonia() retorna la suma de machos y hembras

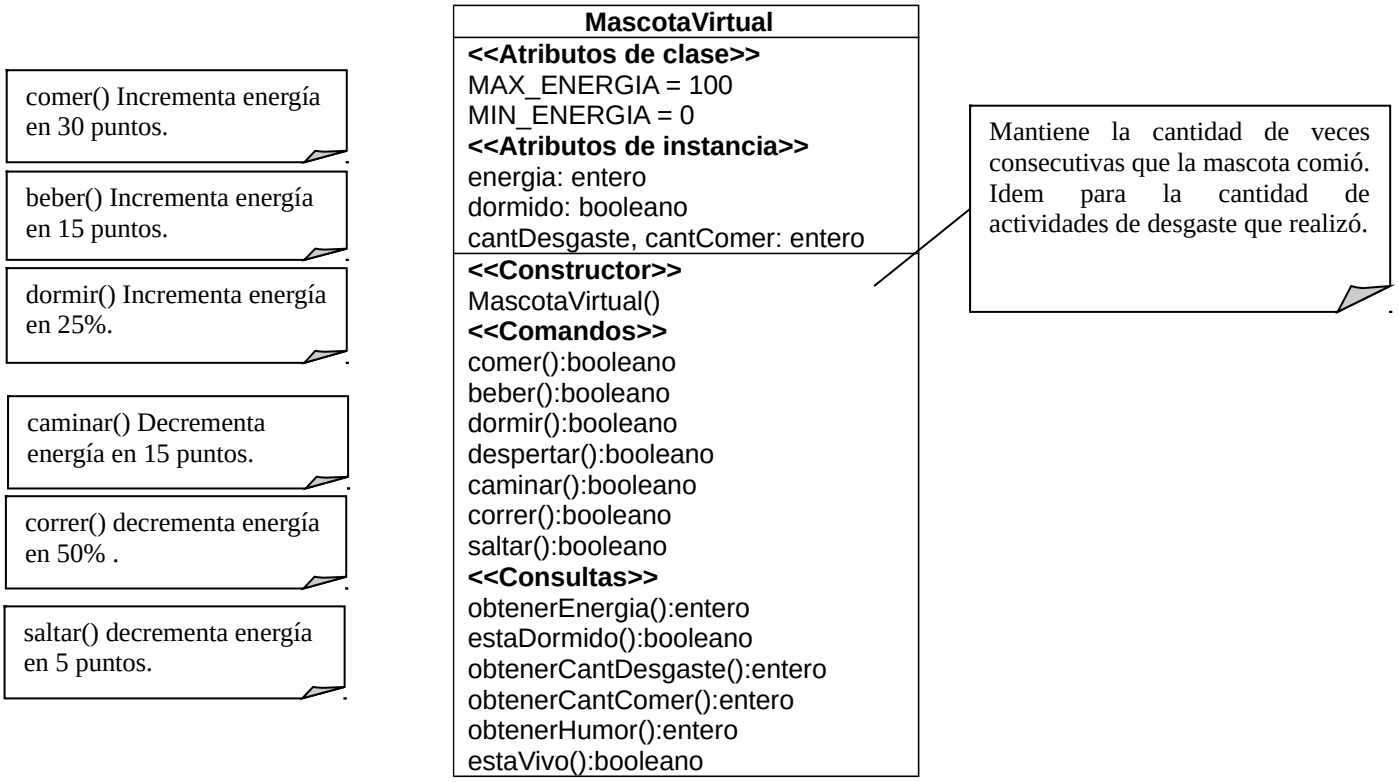
a) Implemente la clase *ColoniaLangostasMutantes* modelada en el diagrama.
b) Implemente una clase Tester que permita verificar los servicios provistos por la clase ColoniaLangostasMutantes.

Ejercicio 5. El *tamagotchi* es un juego donde el objetivo es mantener con vida a una "mascota virtual" la mayor cantidad de tiempo posible. Cada mascota virtual se identifica por medio de su nombre. Pueden realizar diferentes acciones, las que aumentan o disminuyen su nivel de energía, el cual nunca puede disminuir por debajo de 0 ni superar el máximo de 100. Cuando el nivel de energía es cero, la mascota lamentablemente abandona la existencia terrenal. Las operaciones que incrementan el nivel de energía son *comer*, *beber* y *dormir*. Cuando duerme recupera el 25% de su energía. Mientras duerme no puede hacer ninguna otra acción, a menos que antes se lo despierte. Las acciones que decrementan el nivel de energía de la mascota son *caminar*, *correr* y *saltar*, si bien lo hacen en distinto grado. Ninguna mascota puede realizar más de tres acciones de desgaste en forma consecutiva. Al cuarto intento, ignora la orden dada y directamente se pone a dormir. Cuando se vuelva a despertar, podrá volver a realizar a lo sumo tres acciones de desgaste. Asimismo, si come o bebe cinco veces seguidas morirá al instante de indigestión.

La mascota manifiesta diferentes grados de humor, los que expresa mediante un valor numérico que va de 1 a 5, siendo 1 el más infeliz y 5 el más contento. El humor de la mascota depende de su nivel de energía como sigue:

Energía	Humor
0…20	1
21...40	2
41…60	3
61…80	4
81…100	5

La clase MascotaVirtual se modela con le diagrama:



Todos los comandos retornan **true** si se ejecutaron correctamente o **false** en caso contrario (si la mascota está muerta o superó la cantidad de veces que puede realizar la acción de forma consecutiva).

a) Implementar en Java la clase MascotaVirtual con todos los atributos y operaciones necesarias.
b) Implementar una clase Tester que ejecute un ciclo completo de la vida de estas mascotas: nacimiento, actividades varias, muerte.
c) Implementar un segundo Tester que utilice valores aleatorios para decidir qué acciones ejecutar.

Objetivos del Práctico: Interpretar un diagrama de modelado. Ilustrar los conceptos de objeto, atributo de instancia y de clase (solo tipos elementales). Constructores y Métodos. Los comentarios para establecer la funcionalidad de los métodos. La clases String y Random. La clase tester con valores ingresados por el usuario y con valores generados al azar.