Trabajo Práctico 2: Clases y Objetos

Propiedades de los objetos.
Encapsulamiento. Ocultamiento de la Información. Modos de acceso
Métodos como control de acceso.
Acceso a los atributos privados.
El problema del atributo público
Control de acceso por clase.
Métodos getXXX() y setXXX()
Sobrecarga de métodos
Miembros "static" (atributos, métodos)
Constantes. Constantes primitivos. Constantes con referencia a objeto. Cambio del estado del objeto referido.
La relación de asociación: agregación y composición
Diferencias entre agregación y composición
Patrón Singleton

Objetivos de la práctica

Comprender la utilidad del encapsulamiento de datos.
Utilizar getters y setters. Validar datos.
Sobrecarga de métodos
Sobrecarga de constructores
Composición – Agregación – Asociación
Atributos de tipo Static
Arrays de elementos primitivos y de objetos.
Introducción a los Requerimientos.
Aplicación de Inspecciones, recorridos de prueba, checklist.

Ejercicio 1:

Sea la siguiente clase F y la variable *f1,* una instancia de F. ¿Cuál sería el resultado de *f1.p()*?

```
public class F
 {
 private int x = 0;
 private int y = 0;
 public F() { }
 public void p()
 {
 int x = 1;
 System.out.println("x = " + x);
 System.out.println("y = " + y);
 System.out.println("x = " + this.x);
 }
}
```

Ejercicio 2:

Dada la clase Punto y el Main que lo utiliza. Analizar cada qué hace cada instrucción numerada de Main y si alguna provoca error, justificar.

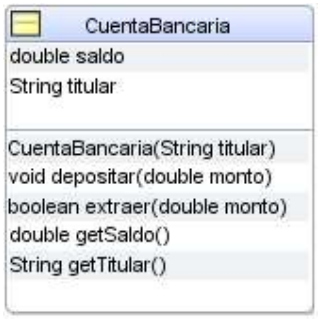
Punto

```
public class Punto
{
private int x;
private int y;
public Punto(intx, inty)
 {
  this.x = x;
  this.y = y;
 }
public void cambia(int x1,int y1)
 {
  setX(x1);
  setY(y1);
 }
public String cartel()
{return"Punto[x="+x+",y="+y+"]";}
}
```

Main

```
   public class PruebaPunto
   {
   public static void main(String[] args)
   {
1  Punto p1 = new Punto(2,3);
2  Punto p2;
3  Punto p3 = newPunto();
4  System.out.println("P1="+p1.cartel());
5  p3=p2;
6  p2=p1;
7  p1.cambia(8,5);
8  System.out.println("P2="+p2.cartel());
   }
   }
```

Ejercicio 3:



Escriba la clase CuentaBancaria, con los atributos y propiedades que se detallan, determinando si serán públicos o privados.

El método extraer devolverá true si la transacción fue exitosa. Y devolverá false, en caso de que el monto que se quiso extraer sea mayor al saldo, en cuyo caso, el saldo no se debe modificar.

Se deben realizar las validaciones necesarias (no se pueden extraer o depositar montos negativos)

Ejercicio 4:

Escribe un programa que represente un pequeño negocio, construye las clases **Empleado**, **Producto** , **Pedido**,y **LineaDePedido** con sus correspondientes getters , setters, así como sus constructores.

Un **Empleado** tendrá un nombre, un teléfono y un Email.

Un **Producto**, tendrá un número de código, una descripción, y un precio unitario.

Un **Pedido** involucra un empleado (al cual es el responsable del pedido), una fecha, y un ArrayList de líneas de pedido cada línea de pedido consta de un producto específico, y de una cantidad. Deberá poder informar el costo total del pedido.

Inciso a)

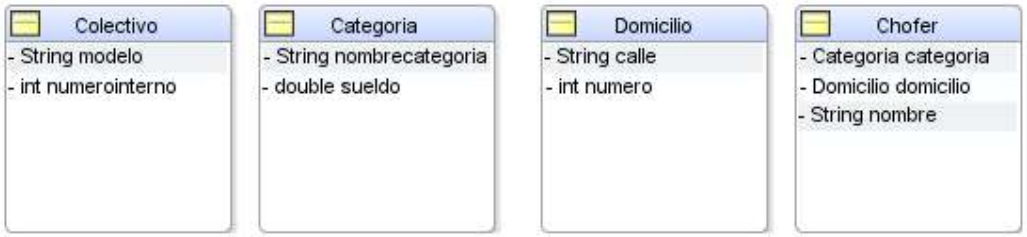
- Realiza el diagrama de clases de las clases involucradas considerando el tipo de relación que las une.

Inciso b)

- Escribe una clase Prueba, que contendrá un método main, en el cual deberás instanciar empleados, y productos, para poder armar un pedido. ¿De quién sería la responsabilidad de conservar la lista de empleados y los pedidos?

Ejercicio 5:

Realiza una aplicación Java que represente una empresa de colectivos.
Deberá modelar las siguientes clases (solo se muestran algunos atributos):



Requerimientos:

1. Cada chofer posee un único domicilio, y tiene una categoría que define su sueldo.
2. El número de interno de un colectivo deberá generarse automáticamente en forma correlativa.
3. Varios choferes pueden tener la misma categoría.
4. A cada chofer se le puede asignar un colectivo.
5. Se deberá poder desvincular un chofer de su colectivo.
6. Un chofer podría no tener un colectivo asignado.
7. Deberá ser posible mostrar la información de cada chofer, incluyendo la información detallada correspondiente al colectivo asignado, o indicar que el chofer no tiene un colectivo asignado (Considerar la posibilidad de sobreescribir los métodos toString).

Agregar una clase Empresa que contenga a las demás.

El sistema debe poder responder a lo siguiente:

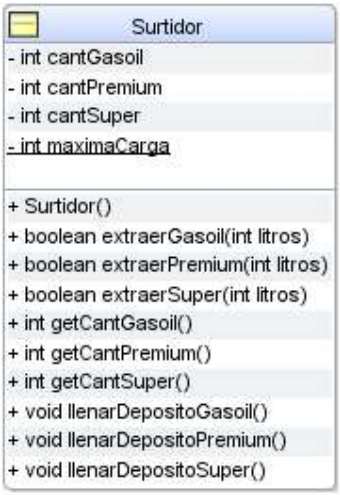
1. ¿Cuántos choferes no tienen un colectivo asignado?
2. ¿Cuántos Colectivos posee la empresa en total?
3. ¿Qué choferes pertenecen a una determinada categoría?
4. ¿Qué categorías tienen un sueldo superior a un monto determinado?
5. ¿Qué choferes tienen un sueldo superior a un monto determinado?

Agregue los atributos y métodos que crea convenientes.

Realice el diagrama UML diferenciando las relaciones entre las clases.

Escribe una clase Prueba, en cuyo método main, se instancie una clase Empresa, en la cual se agregarán varios choferes con diferentes categorías, y colectivos para comprobar la funcionalidad del sistema. Comprueba de vincular colectivos a choferes y mostrar la información de ellos.

Ejercicio 6:



Una estación de servicio cuenta con surtidores de combustible capaces de proveer Gasoil, Nafta Súper 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 el 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. Los métodos retornan true si se ha podido cargar la cantidad requerida, y false, en caso contrario. Cuando se repone un combustible en el surtidor, se llena el depósito completo de ese combustible.

a) Implementa la clase Estación, que contendrá varios surtidores.

(el diagrama no esta completo, se deberán agregar atributos y métodos)

El sistema deberá ser capaz de satisfacer las siguientes necesidades:

1. Conocer la cantidad total de surtidores
2. Conocer la existencia en litros de un determinado tipo de combustible.
3. Informar cual es el surtidor que ha realizado mayor cantidad de ventas de un tipo de combustible.
4. Conocer el histórico de los litros vendidos de un determinado combustible, de un determinado surtidor y de toda la estación.

b) Escriba una clase Prueba con un método main() que permita verificar los servicios provistos.

Ejercicio 7:

Escribe las clases necesarias para poder manejar una agenda personal que permita registrar los teléfonos de nuestros contactos.

Los datos de los contactos que se deben guardar son nombre, teléfono fijo y varios teléfonos celulares.

Se sugiere el uso de Hashmap.

El sistema debe satisfacer las siguientes condiciones (esto generará la lista de requerimientos funcionales)

1. ABM de contactos
2. Búsqueda de un contacto por nombre
3. Evitar contactos con el mismo nombre
4. Mostrar todos los contactos

Ejercicio 8: Fútbol de primera división

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, una colección de Jugadores, una cantidad de partidos ganados, otra de empatados y otra de perdidos, una cantidad de goles a favor y otra de goles en contra. Un jugador tiene un nombre, un año de nacimiento, 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. Un partido tiene dos equipos (que no pueden ser el mismo), y una fecha.

Para un jugador se desea calcular el promedio de goles de un jugador 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 goleador 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 se consideran los goles en contra. Si hay coincidencia se devuelve uno cualquiera.

Para un partido, se desea poder obtener el detalle de los goles anotados indicando que jugador lo convirtió, y el resultado final del partido.

1. Implementar las clases necesarias.
2. Implemente el programa para probar las clases Implementadas

(Para evitar la doble referencia entre los equipos y el torneo, analizar el patrón Singleton)

Ejercicio 9: Fútbol amateur entre amigos

(Este ejercicio presenta un enfoque muy diferente al anterior, identifica previamente las clases involucradas y analiza las responsabilidades de cada una)

Un grupo de amigos se reúnen todos los miércoles para jugar al fútbol de salón. Cada miércoles, arman dos equipos diferentes para mezclarse, e ir jugando todos con todos, pero al mismo tiempo, como no todos tienen la misma habilidad, se desea armar los equipos de forma equilibrada. Para ayudar a armar los equipos y llevar ciertas estadísticas se desea desarrollar la siguiente aplicación que pueda responder a lo siguiente:

- De cada jugador se desea conocer su nombre y su puntaje. (por cada partido ganado se le suman 3 puntos, y por cada empate, 1 punto)

- En cada fecha se debe poder armar dos equipos (agregar y quitar jugadores de cada uno) y conocerse el puntaje de cada equipo igual a la suma de los puntajes de sus participantes. Analizar posibles restricciones (un jugador no puede estar en ambos equipos, un equipo debe tener al menos 4 jugadores, los equipos no necesariamente deben tener igual cantidad de jugadores, etc.)
- Una vez armados los equipos se debe poder ingresar el resultado del encuentro para actualizar las estadísticas de los participantes.
- Se debe poder consultar un histórico de los encuentros indicando la fecha del partido, los participantes y el resultado.
- De cada jugador se debe poder consultar la cantidad de partidos jugados, empatados, y perdidos
- Dados dos jugadores se quiere consultar cuántos partidos jugaron en el mismo equipo y cuantos enfrentados.

Nota: Para crear una fecha en particular se debe usar el objeto GregorianCalendar, por ejemplo para representar el 20 de marzo de 2019 se usa:

```
GregorianCalendar fecha=new GregorianCalendar(2019,2,20);
```

El constructor de con tres parámetros enteros crea una fecha con el año, mes y día especificado, los meses se inician en 0 por eso marzo se corresponde con el entero 2.
El constructor sin parámetros crea la fecha actual del sistema.
Para poder convertir a String una fecha se utiliza un objeto de tipo SimpleDateFormat, por ejemplo:

```
SimpleDateFormat sdf =
 new SimpleDateFormat("EEEEE dd 'de' MMMMMMMMM 'de' yyyy");

System.out.println(sdf.format(fecha.getTime()));
```

El String pasado como parámetro en el constructor del SimpleDateFormat sirve para configurar la máscara de formato de fecha que se desea usar. Consulta la documentación para ver mas opciones que permiten visualizar hora, minutos, segundos, etc.