	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	TISTA GARCÍA EDGAR
Asignatura:	PROGRAMACIÓN ORIENTADA A OBJETOS
Grupo:	3
No de Práctica(s):	PRACTICA #5-6 ABSTRACCION Y ENCAPSULAMIENTO & ORGANIZACIÓN DE CLASES
Integrante(s):	CARRILLO CERVANTES IVETTE ALEJANDRA
No. de Equipo de cómputo empleado:	TRABAJO EN CASA
No. de Lista o Brigada:	08
Semestre:	2022 - 1
Fecha de entrega:	27 OCTUBRE 2021
Observaciones:	

CALIFICACIÓN: _____

Práctica #5-6. Abstracción y Encapsulamiento & Organización de clases

Objetivo General.
(P5) Aplicar el concepto de abstracción para el diseño de clases que integran una solución, utilizando el encapsulamiento para proteger la información y ocultar la implementación.
(P6) Organizar adecuadamente las clases según su funcionalidad o propósito bajo un namespace o paquete.

Objetivo de clase.
Realizar un programa orientado a objetos que permita aplicar los conceptos más importantes de los modificadores de acceso, uso de paquetes y composición de clases.

ACTIVIDADES PARA EL DESARROLLO DE LA PRÁCTICA

Ejercicio1. Composición de clases

Este ejercicio tiene como función principal crear un programa para la administración de un ejército, para lo cual se realizaron tres clases: "Militar", "Batallón" y "División". En cada clase se realizaron los métodos de acceso "getters" correspondientes a cada atributo que se les asigno, además se crearon los métodos constructores que se utilizarían. A continuación, se describe el principio de cada una de las clases:

División

Tiene como atributos:

- Nombre de la división. Variable tipo String (Águila, Jaguar, Tigre, Halcón, …)
- Zona. Variable tipo String (norte, sur, este, suroeste, …)
- Lista de batallones. Lista tipo Batallón. En esta lista se agregarán los batallones creados en cada división.
- Divisiones. Variable de clase tipo Int, la cual se inicializa en cero.

Tiene un método constructor vació y un método constructor donde se inicializarán todos sus atributos, en ambos casos la variable de clase irá aumentando cada vez que se usen con el fin de llevar la cuenta de cuantas divisiones se han creado con forme avanza el programa.

Batallón

Tiene como atributos:

- Identificador del batallón. Variable tipo int (1234, 4567, 4567, …)
- Categoría. Variable tipo String. (infantería, caballería, artillería)
- Ubicación. Variable tipo String. (Estado de la República)
- Lista de militares. Lista tipo Militar. En esta lista se agregarán los militares creados en cada batallón.
- Batallones. Variable de clase tipo Int, la cual se inicializa en cero.

Tiene un método constructor vació y un método constructor donde se inicializarán todos sus atributos. Al igual que la clase anterior, en ambos casos la variable de clase irá aumentando cada vez que se usen con el fin de llevar la cuenta de cuantos batallones se han creado con forme avanza el programa.

Militar

Tiene como atributos:

- Matricula. Variable tipo int (1234, 4567, 9635)
- Nombre. Variable tipo String (nombre del militar)
- Grado militar. Variable tipo String (general, teniente, capitán)

Al igual que las clases descriptas previamente, tiene un método constructor vació y un método constructor donde se inicializarán todos sus atributos y en cada método constructor la variable de clase irá aumentando cada vez que se usen con el fin de llevar la cuenta de cuantos militares de han creado conforme avanza el programa

Principal

Para probar el funcionamiento de este programa se creó una clase ***Principal.java***, la cual contendrá un menú con las siguientes opciones:

```
------------------------------------------------------------

                        MENÚ

  1. Crear División con Batallones y Militares
  2. Ver totales
  3. Mostrar todas las Divisiones
  4. Salir
-> Elige una opción:
```

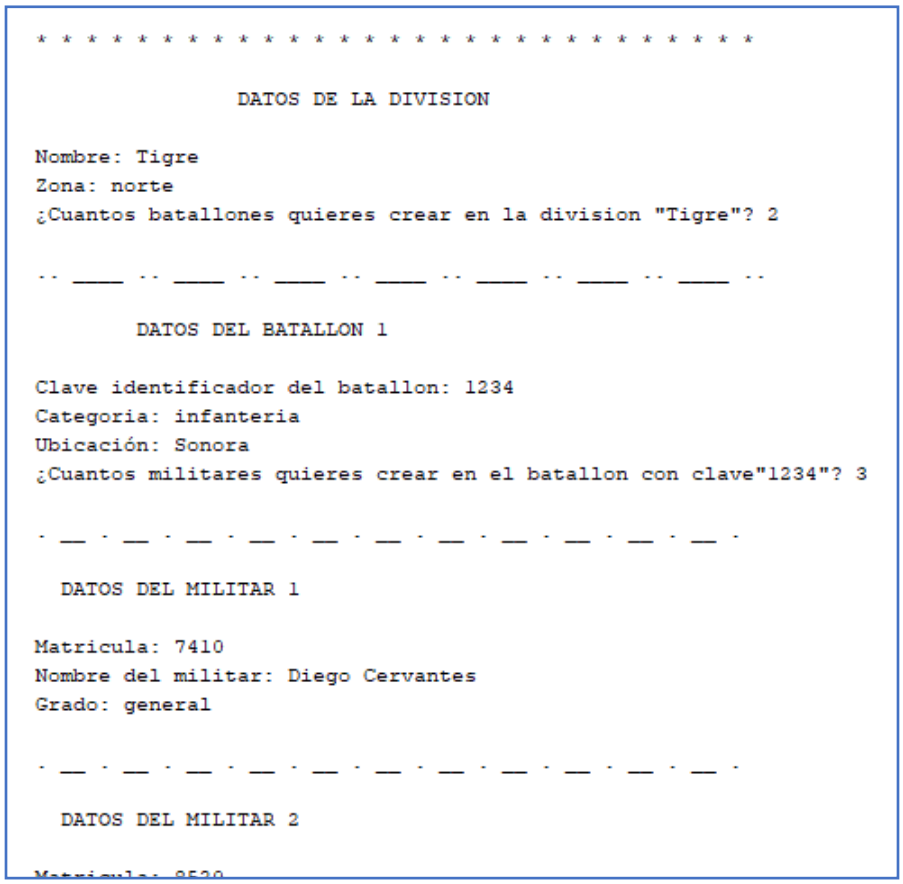
Menú de opciones -> Principal.java

→ Primera opción

Para la implementación de la creación de una división con batallones y militares, se hizo uso de tres listas: la que tiene como atributo la clase División (la cual contiene batallones), la que tiene como atributo la clase Batallón (la cual contiene militares) y una nueva que se crea en la clase principal, la cual contendrá las Divisiones. Primero, siguiendo la metodología Top-Down, se le pidió al usuario que ingresará los datos de la división que crearía y cuantos batallones quisiera que tuviera dicha división, con base a ese dato se realizó un ciclo for con la finalidad de que se le pidiera al usuario la información de batallones el número de veces necesarias para ser creados, dentro de este for se le pidió al usuario el número de militares que quería que tuviera cada batallón, y se realizó otro ciclo for, con el fin de que se le pidiera al usuario la información de cada militar el número de veces correspondiente.

Cabe mencionar que cada vez que se creaba una división, un batallón o bien, un militar se agregaba a su lista correspondiente y si era necesario (en el caso de división y batallón) se le pasaba esta lista como parámetro al método constructor.

La salida es la siguiente:

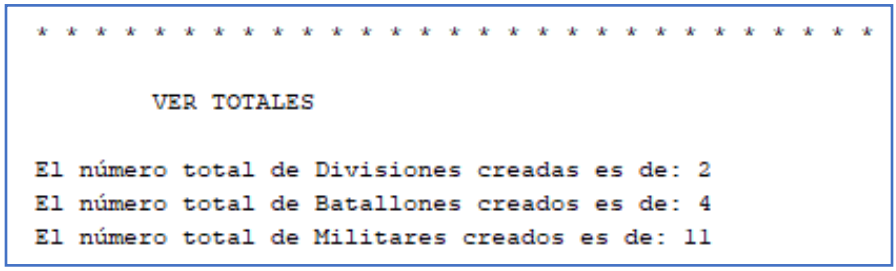


Nota: Para que no sea una captura de pantalla muy larga de cuando el programa le pide dato por dato al usuario, se recortó.

Crear divisiones -> Principal.java

→ *Segunda opción*

Esta opción tiene como objetivo informarle al usuario cuantas divisiones, batallones y militares se han creado desde que ejecuto el programa. Para lograr su implementación, se creó un método en cada clase, este cual retornaría el valor de la variable estática que se creó al principio del programa la cual lleva la cuenta del número total de Divisiones, Batallones y Militares se han creado. Se llamó cada método desde la clase Principal y su salida es la siguiente:



Ver totales -> Principal.java

→ *Tercera opción*

Esta opción tiene como objetivo, imprimir todas las divisiones con sus respectivos batallones y militares creados, esto se logró haciendo un método en cada clase, este método obtendrá la información de cada División, Batallón y Militares respectivamente, mediante el método de acceso "get". En la clase principal, en la opción correspondiente, se realizó una estructura condicional, la cual indica que, si la lista "Divisiones" en donde se encuentran todas las divisiones creadas esta vacía, entonces no habría ninguna división por mostrar, por ende, tampoco habría algún batallón o militar y mostraría lo siguiente en pantalla:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        VER DIVISIONES

NO HAY DIVISIONES CREADAS ):
```

Ver divisiones (Cuando no hay ninguna) -> Principal.java

De lo contrario, si la lista donde se almacenan las divisiones tiene algún elemento, se recorrerá un ciclo for el cual llama al método "imprimirDivision" que este llama a su vez a "imprimirBatallon" y este a su vez "imprimirMilitares" de sus clases correspondientes. La salida de esta implementación es la siguiente:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        VER DIVISIONES

_____________________________________________________

        DATOS DE LA DIVISION

  Nombre: Tigre
  Zona: norte
__  __  __  __  __  __  __  __  __  __  __  __  __  __

        DATOS DEL BATALLON 1

  Clave identificador: 1234
  Categoria: infanteria
  Ubicación: Sonora
. __ . __ . __ . __ . __ . __ . __ . __ . __ .

    MILITAR 1
  Matricula: 7410
  Nombre del militar: Diego Cervantes
  Grado militar: general
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .

    MILITAR 2
  Matricula: 8520
  Nombre del militar: Luis Duran
  Grado militar: teniente
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

MILITAR 3
Matricula: 9632
Nombre del militar: Julio Flores
Grado militar: teniente

DATOS DEL BATALLON 2

Clave identificador: 5678
Categoria: caballeria
Ubicación: Coahuila

MILITAR 1
Matricula: 7894
Nombre del militar: Santiago Sanchez
Grado militar: teniente

MILITAR 2
Matricula: 4561
Nombre del militar: Gael D□az
Grado militar: general

MILITAR 3
Matricula: 3210
Nombre del militar: Cesar Carrillo
Grado militar: capitan

DATOS DE LA DIVISION

Nombre: Puma
Zona: Sur

DATOS DEL BATALLON 1

Clave identificador: 7894
Categoria: artilleria
Ubicación: Oaxaca

MILITAR 1
Matricula: 7532
Nombre del militar: Alejandro Arteaga
Grado militar: capitan

```
   MILITAR 2
  Matricula: 8642
  Nombre del militar: Andres Pe□a
  Grado militar: general
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .

   MILITAR 3
  Matricula: 1086
  Nombre del militar: Jaret Santiago
  Grado militar: teniente
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .

__  __  __  __  __  __  __  __  __  __  __  __  __  __

        DATOS DEL BATALLON 2

  Clave identificador: 9635
  Categoria: caballeria
  Ubicación: Guerrero
. __ . __ . __ . __ . __ . __ . __ . __ . __ .

   MILITAR 1
   MILITAR 1
  Matricula: 9512
  Nombre del militar: Yahir Duran
  Grado militar: general
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .

   MILITAR 2
  Matricula: 1234
  Nombre del militar: Carlos Aldrade
  Grado militar: teniente
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .

   MILITAR 3
  Matricula: 1606
  Nombre del militar: Edwin D□az
  Grado militar: capitan
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

Ver divisiones (Cuando si hay divisiones creadas) -> Principal.java

Ejercicio 2. Modificadores de acceso

Este programa tiene como función principal llevar los registros del personal de una empresa, para llevar a cabo esto, se crearon 4 clases diferentes, una de ellas es una clase padre de la cual heredan las otras 3. Más adelante se describirá cada una de las clases mencionadas. El diagrama de la herencia que se presenta es el siguiente:

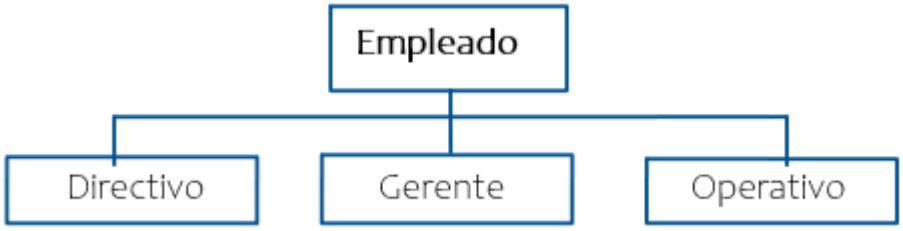
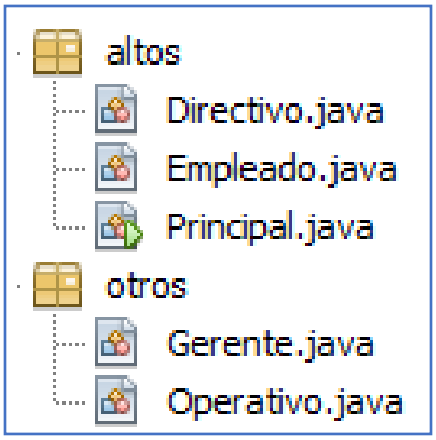


Diagrama Herencia

Estas clases estan divididas de la siguiente manera:



package

En el package **altos** se encuentran la clase directivos, empleado y la clase principal; mientras que en el package **otros** se encuentran la clase gerente y operativo.

Clase Empleado

La clase empleado es una clase publica, la cual se toma como clase padre o superclase, en esta se tienen los siguientes atributos de cada empleado creado posteriormente:

- Nombre
- Apellido
- salarioBase
- Puesto

Esta clase tiene un constructor donde se inicializarán el nombre y el apellido de cada empleado al momento que lo manden a llamar; también, se tienen diferentes métodos de acceso correspondientes a cada uno de los atributos de esta clase. Dichos métodos son los siguientes:

→ getsalarioBase()
Es un método de acceso para saber el salario de una persona, es público para que todas las personas puedan ver su salario.

→ setSalarioBase()
Es un método de acceso para modificar el salarioBase de algún empleado, este método es de tipo friendly para que solo los que estan dentro del package(directivo) lo puedan usar.

→ setPuesto()
Es un método de acceso para modificar el puesto de una persona, es de tipo Protected con el fin de que solo la misma clase u otra derivada de esa pueda acceder a este método.

→ getPuesto()
Al igual que el método anterior es tipo protected con el fin de que solo la misma clase pueda modificarla u otra derivada de esta.

Clase Directivo

Esta clase es una clase que se deriva de Empleado, contiene como atributo una variable de tipo private la cual indica el salarioAdicional. Llama al método constructor de su clase padre (Empleado) con ayuda de la palabra reservada “Super”, con el fin de solo inicializar el nombre y el apellido del empleado, en este caso directivo.
Tiene los siguientes atributos:

→ setSalarioAdicional()
Es un método de acceso friendly para que solo los que estan dentro del package, puedan modificar a este dato.

→ getSalarioAdicional()
Es un método de acceso friendly para que solo los que estan dentro del package, puedan acceder a este dato.

→ setSalarioOtro()
Es igual que los métodos anteriores de tipo friendly para que solo los que están dentro del package puedan modificar este otro salario. Se puede observar que se manda a llamar al método setSalarioBase de la clase padre (Empleado).

→ getSalarioOtro()
Al igual que los métodos anteriores solamente pueden ver el salarioOtro del empleado los que se encuentran dentro del package, ya que es tipo friendly. Al igual que el método anterior, se observa que se llama al método getsalarioBase de la clase padre.

Clase Gerente

Esta clase también se deriva de Empleado, contiene como atributo el área y el numero de subordinados que tiene el empleado. Llama al método constructor de su clase padre (Empleado) con ayuda de la palabra reservada “Super”, con el fin de solo inicializar el nombre y el apellido del empleado, en este caso gerente.
Tiene los siguientes atributos de tipo public:

→ getArea()
Retorna el área donde se encuentra el gerente.

→ setArea()
Se puede modificar el area del gerente.

→ getNumSubordinados()
Cualquier clase puede saber el número de subordinados puede tener el gerente.

→ setNumSubordinados()
Cualquier clase puede cambiar el número de subordinados que puede tener el gerente.

→ getSalarioSubordinado()
Cualquier clase puede acceder al Salario del subordinado. Se observa que en este método se manda a llamar a un método de la clase padre.

Clase Operativo

Es una clase derivada de la clase padre (Empleado), tiene como atributos la el turno y la jornada laboral del empleado, al igual que las clases anteriores tiene el método constructor del padre y tiene los siguientes métodos públicos:

→ getTurno()
→ setTurno()
→ getJornadaLaboral
→ setJornadaLaboral

Principal

En la clase principal, se ponen a prueba cada método de acceso de cada clase, se utiliza el método constructor de cada clase según el tipo de objeto que se crea, al igual que se le asigna el puesto, salario, turno etc. respectivamente. Con el método set el directivo puede cambiar el salario de gerentes y operativos. La salida es la siguiente:

```
run:
desde operativo 1 5000.0
desde operativo2  5500.0
desde gerente, salario gerente 20000.0
desde gerente salario operativo 5000.0
desde directivo, salario gerente20000.0
desde directivo, salario operaivo 5500.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Empleado -> Principal.java

Conclusiones

Se cumplieron los objetivos de esta práctica, ya que en el primer ejercicio se realizó un programa en el cual se vio la composición de clases, en donde se implementaron objetos de objetos; mientras que, en el segundo programa se vió más a detalle como funcionan los package y los métodos de acceso “setters”. Personalmente, siento que el primer ejercicio puede haber implementado de una mejor manera los métodos de acceso, ya que solo utilice el método get para obtener los atributos de cada objeto que cree; sin embargo, considero que se logró cumplir el propósito de ese ejercicio ya que fue la salida correcta en el programa. Debo destacar que me sirvió de guía la práctica parecida que se realizó en Estructura de Datos y Algoritmos I, para la creación de los atributos, ya que no se me ocurrían muchos.

En cuanto al segundo ejercicio, considero que fue un muy buen ejemplo para trabajar con métodos de acceso setters, así como el encapsulamiento, ya que no todos los package podían acceder a algunos atributos de algunas clases.

Y aunque no describí mucho a detalle el segundo ejercicio debido a que algunas cosas todavía me confunden con respecto a como acceder a diferentes clases, considero que los ejercicios para esta práctica fueron muy buenos para conocer estos temas más a fondo.