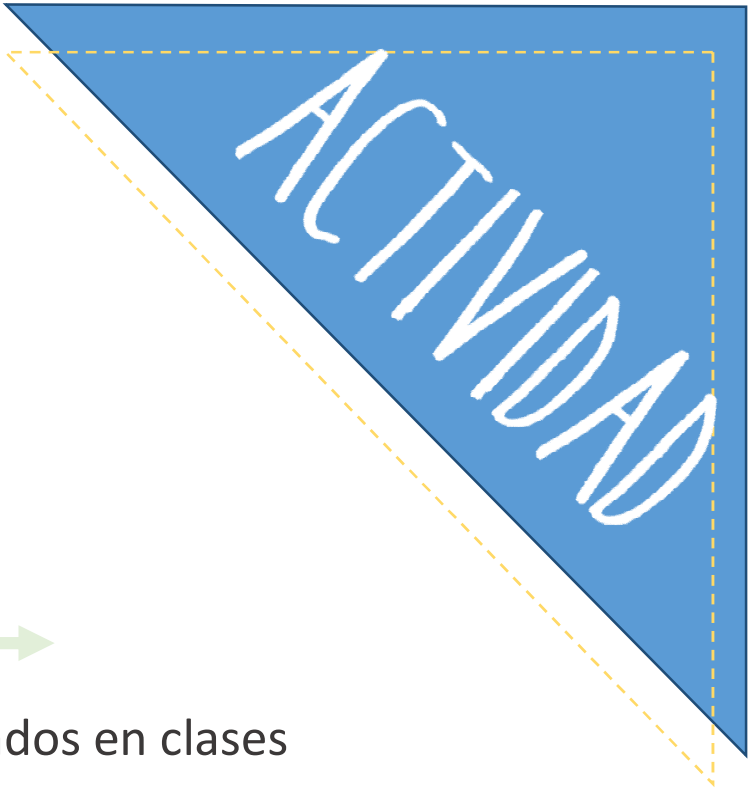

Desarrollo de programas organizados en clases

Desarrollo de Aplicaciones Web/Desarrollo de Aplicaciones Multiplataforma

Programación

Linkia FP

Actividad

Desarrollo de programas organizados en clases

Objetivos

- Crear clases e instanciar objetos
- Definir atributos y métodos.
- Crear y utilizar constructores
- Utilizar los atributos y los métodos de los objetos
- Crear y utilizar clases heredadas, interfaces y clases abstractas.
- Crear y utilizar atributos estáticos.
- Crear clases heredadas que sobrecarguen la implementación de métodos de la superclase.
- Realizar programas que implementen y utilicen jerarquías de clases.

¿Cómo lo hago?

1. Rellena los datos que se piden en la tabla“Antes de empezar”.
2. Haz uso de fuentes comunes como Arial, Calibri, Times New Roman etc.
3. Utiliza el color negro para desarrollar tus respuestas y usa otros colores para destacar contenidos o palabras que creas necesario resaltar.
4. Entrega un zip que contenga todos los archivos. java que has creado. Para poder aprobar un ejercicio, éste debe poder ejecutarse sin errores.
5. Recuerda nombrar el archivo zip siguiendo estas indicaciones:
    - Ciclo_Módulo o crédito_Tema_ACT_númeroactividad_Nombre y apellido
        - Ejemplo: AF_M01_T01_ACT_01_Maria Garcia

Antes de empezar…	
Nombre	GARA
Apellidos	GONZÁLEZ SOSA
Módulo/Crédito	M03 PROGRAMACIÓN
UF (solo ciclos LOE)	UF4
Título de la actividad	Desarrollo de programas organizados en clases

Se debe entregar un zip que contenga todos los archivos. java que has creado. Para poder aprobar un ejercicio, éste debe poder ejecutarse sin errores. Crea los archivos .java dentro de una carpeta de nombre actividad05

Actividad 01

Cuenta Corriente

1. Crea el package**actividad05.ejercicio01. cuentaCorriente**con una clase ***Utilidades*** que contendrá dos métodos para pedir datos al usuario:
   - pedirEntero. Recibe una pregunta como parámetro, la muestra por consola y pide un dato al usuario hasta que el usuario introduzca un valor entero válido.
   - pedirDecimal. Recibe una pregunta como parámetro, la muestra por consola y pide un dato al usuario hasta que el usuario introduzca un valor doble válido.

Creamos los packages indicados y rescatamos los ficheros pedirEntero y pedirDouble que ya teníamos preparados en ejercicios anteriores.

2. Añade al package una clase ***Cuenta***, que tendrán los siguientes atributos: un número de cuenta que las identifica, un saldo, y el nombre del titular. La clase debe incorporar un constructor que reciba como parámetros todos los atributos. Implementar, además, los métodos siguientes:
   - getNumeroCuenta. Devuelve el número de la cuenta.
   - getSaldo. Devuelve el saldo de la cuenta.
   - getTitular. Devuelve el nombre del titular de la cuenta.
   - setSaldo. Establece el saldo de la cuenta.
   - setTitular. Establece el nombre del titular de la cuenta.

Creamos la Clase Cuenta y creamos sus atributos de clase (numCuenta, saldo, nombre). Los creamos en private así que tendremos que acceder a éstos, cuando estemos fuera de la clase, mediante los getters and setters.

Inicializamos los atributos en el constructor de la clase y añadimos los getters and setters. En la clase también añadí la función mostrarInfo para que luego al solicitar el estado de los atributos no tuviera que llamar a cada atributo uno por uno.

Se deberán utilizar las funciones *pedirEntero* y *pedirDecimal*definidas en la clase *Utilidades* siempre que se quiera pedir un valor entero o un valor doble.

3. Añade al package una clase **TestCuenta** con el método main en el que se creen al menos dos cuentas. Crear un menú principal con las siguientes opciones teniendo presente que **con cada operación se debe mostrar el estado actual de todas las cuentas**:
   - Consultar saldo: ha de preguntar la cuenta y mostrar el saldo que tiene.
   - Ingresar dinero: ha de preguntar la cuenta e añadir el saldo a la cuenta correspondiente.
   - Sacar dinero: ha de preguntar la cuenta, el saldo a sacar y mostrar el saldo restante.
   - Realizar transferencia: ha de sacar el dinero de una cuenta para meterlo en otra.

En la Clase TestCuenta creamos el main.
Aquí creamos los objetos de la clase Cuenta. Los hago en un Array con dos posiciones.
Hice el menú y cada case del menú llama su método **correspondiente y todos empiezan llamando a mostrarInfo para ver el estado de los atributos (núm. Cuenta, saldo y nombre.**

1) mostrarCuenta: se recorre el array y en cada posición va acceder desde el objeto cuenta de cada posición a mostrarInfo y mostrará el estado actual de sus respectivos atributos de clase. Función void puesto que solo va a mostrar el estado por pantalla.

2) consultarSaldo: llamamos a pideEntero para que el usuario elija la posición de la cuenta que quiere consultar y se muestra mediante el objeto cuenta con dicha posición y la función get, ya que los atributos de clase eran privados.

3) ingresarSaldo: Utilizamos pideEntero y pideDecimal para pedir al usuario la posición de la cuenta a ingresar y la cuantía. Luego con la función getSaldo obtenemos el saldo actual del objeto seleccionado y hacemos uso del setSaldo para pasarle el saldo actual + el ingreso realizado a dicho objeto Mostramos por pantalla la operación realizada.

4) sacarDinero: Utilizamos pideEntero y pideDecimal para pedir al usuario la posición de la cuenta a retirar y la cuantía. Luego con la función getSaldo obtenemos el saldo actual del objeto seleccionado y hacemos uso del setSaldo para pasarle el saldo actual – el importe a retirar a dicho objeto Mostramos por pantalla la operación realizada.

5) Transferencia: Pedimos al usuario que seleccione la posición de la cuenta de origen, la de destino y la cantidad de dinero a transferir. Luego con las funciones getSaldo y setSaldo llamamos al atributo Saldo de la cuenta de origen y a el atributo Saldo de la cuenta de destino y le ingresamos/retiramos el importe reseñado. Mostramos por pantalla la operación realizada.

Actividad 02

Heladeria

1. En el package**actividad05.ejercicio02** define una interface**Vendible** que contenga los métodos:
    a. getPrecio()
    b. setPrecio(double precio)

La interfaz Vendible solo va a tener las definiciones de los métodos que van a heredar las nuevas clases que implementen de ellas.

2. Añade al package una clase **Loteria** que implemente Vendible y que estructure la siguiente información:

a. Precio
b. Número de lotería

Sobrescribe toString para que retorne una string con la información del número.

La clase Lotería hereda/implementa la interfaz Vendible por lo tanto contiene todos los métodos que contenga Vendible.
Declaramos privados los atributos precio y numeroLotería, creamos el constructor para iniciarlos y codificamos los getters y setters.
Con el @override sobreescribimos el método toString para poder mostrar fácilmente los atributos de la Clase Lotería.

3. Añade al package una clase Comida de tipo abstracta y que implemente Vendible. Debe permitir estructurar la siguiente información de una comida:
   a. una variable estática que almacene el tipo de datos que estructura la clase. Por defecto su valor ha de ser “comida”
   b. Nombre
   c. precio

   Añade como mínimo los siguientes métodos:
   d. Getters/Setters para los atributos
   e. Sobrescribe toString para que retorne una string con la información de la comida.

Clase Comida Clase abstracta de la que no se puede crear objetos sino heredarla desde clases hijas. Esta Clase Comida hereda/implementa la interfaz Vendible por lo tanto contiene todos los métodos que contenga Vendible.

Declaramos los atributos de la clase protegidos para poder acceder a ellas solo desde otras clases hijas, sin recurrir a los getters y setters.

Creamos los constructores donde se inicializa los atributos y para el "tipo" pone un valor por defecto "comida". Codificamos los getters y setters necesarios y con el @override sobreescribimos el método toString para poder mostrar fácilmente los atributos de la Clase Comida.

4. Añade al package una clase Horchata que herede de Comida y que permita estructurar la siguiente información:
   a. cantidad
   b. % de chufa

   Añade como mínimo los siguientes métodos:
   c. Getters/Setters para los atributos
   d. Constructor que establezca los atributos anteriores.
   e. Sobrescribe toString para que retorne una string con la información de la comida.

Clase Horchata hereda/extiende de la Clase abstracta Comida por lo tanto contiene todos los métodos y atributos que contenga Comida y Vendible.
Declaramos los atributos de la clase (Kcal, cantidad, porcentajeChufa y los inicializamos en el constructor. Desde el constructor tenemos que llamar al constructor de la Clase Padre (Comida) que ya consta definidos) con el súper constructor.

5. Añade al package una clase **Cucurucho** que herede de Comida y que permita estructurar la siguiente información:
    a. Un array de SaborHelado, para almacenar los sabores del cucurucho.
    b. Constructor que establezca el nombre, precio base , kcal y número de bolas de helado que puede tener como máximo el cucurucho.

    Añade como mínimo los siguientes métodos:

    c. Debe llamar al superconstructor correspondiente para establecer que el tipo es "cucurucho"
    d. Sobrescribe toString para que retorne una string con la información del cucurucho y de cada una de las bolas de helado que tiene.
    e. addBolaHelado() debe recibir por parámetro una bola de helado , en qué posición de las bolas de helado que tiene el cucurucho se quiere añadir y añadirla si la posición es correcta.
    f. calculaPrecioTotal() retorna la suma del precio base del cucurucho más el de todas las bolas de helado que tenga.

    La Clase Cucurucho hereda/extiende de la clase abstracta Comida por lo tanto contiene todos los métodos y atributos que contenga Comida y Vendible. Declaramos los atributos de la Clase Cucurucho (numeroBolas y creamos un Array de Clase Helado para poder mostrar los atributos pedidos en el punto 6 y 7 del ejercicio(aparte del sabor en el ejemplo se ve el tipo de edulcorante y el % de grasa). Atributos propios de las bolas de helado.

    Con el método addBolaHelado, que recibe todas las variables necesarias para poder hacer la bola de helado (Clase Helado) + la posición donde se va almacenar, creamos el objeto bola de helado (HELADO). Obligando a que exista esa posición (>=0 y la posición introducida debe ser menos que el tamaño del array existente (numeroBolas).

    Hacemos un @Override para sobreescribir el getPrecio de la Clase padre para que se calcule el precio del cucurucho con el precio de las bolas de helado. Con el @override sobreescribimos el método toString para poder mostrar fácilmente los atributos de la Clase Cucurucho.

6. Añade al package una clase Heladeria con el método main que inicialmente contenga un array de nombre “pedido” con los siguientes valores:
    a. El numero de loteria "119383939" de 3.5€

b. Una horchata de 250ml, 20kcal, 30% de chufa y 2.5€.
c. Un cucurucho de galleta de 20kcal y precio 1€. El cucurucho tiene dos sabores:
    i. de vainilla de 30kcal , 1€ , 15% grasa con azúcar
    ii. de chocolate de 15kcal, 1€ , 15% grasa con aspartamo.
d. Un cucurucho de galleta de chocolate de 25kcal y precio 1.5€. El cucurucho tiene dos sabores:
    i. de cookies de 35kcal , 1€ , 20% grasa con azúcar
    ii. de fresa de 10 kcal, 1€ , 5% grasa con aspartamo.

En la Clase Heladería realizamos el main. El array "pedido" va a ser de tipo de la interfaz "Vendible" puesto que es el único contenedor que puede contenerlo todo.
Inicializamos el array "pedido" de tipo "Vendible" con los 4 pedidos del ejemplo.
Creamos en cada posición del array el objeto requerido pasándoles por parámetros a sus respectivos constructores las variables requeridas. Cuando llegamos al pedido [2] y [3] Cucurucho, al incluir también las bolas de helado tenemos que "forzar" al array a hacer dos posiciones más en cada posición inicial para que contengan los sabores.

7. Muestra la información de todos los productos que forman el "pedido" por consola con un formato parecido al siguiente:

```
Información del pedido:
0-Loteria [numero=119383939, precio=3.5]
1- nombre: horchata, 20.0kcal, precio:2.5e, cantidad:250.0, chufa:30.0%
2- nombre: chocolate, 20.0kcal, precio:1.0e
     bola0- nombre: vainilla, 30.0kcal, precio:1.0e grasa:15.0%, edulco
     bola1- nombre: chocolate, 15.0kcal, precio:1.0e grasa:15.0%, edulc
3- nombre: galleta, 25.0kcal, precio:1.5e
     bola0- nombre: cookies, 35.0kcal, precio:1.25e grasa:20.0%, edulco
```

En verPedido, recorremos el array "pedido" para que lo vaya mostrando todo y vamos sumando los precios de los pedidos para conocer el total.

8. Muestra un menú al dependiente para que hasta que no elija salir, pueda ver el pedido actual y/o sustituir tantas veces como se desee un elemento del pedido por una horchata con los valores que él indique.

   A continuación, se muestra un ejemplo de ejecución:

```
Quieres hacer alguna otra operación?
 1-Sustituye un pedido por una orchata
 2- Ver el pedido actual
 0- Salir
1
Indica el número de producto del pedido a modificar
3
Nombre de la horchata?:
Rica horchata
Precio?:
33
KCal?:
10
Cantidad de horchata?:
20
Qué porcentaje de chufa tiene?:
80
Quieres hacer alguna otra operación?
 1-Sustituye un pedido por una orchata
 2- Ver el pedido actual
 0- Salir
2
Información del pedido:
```

En modificarPedido le pedimos al usuario que elija el valor de las variables de la horchata y creamos una nueva horchata con las variables elegidas.