PROGRAMACIÓN ORIENTADA A OBJETOS – UNALMED 2024-2

Preguntas de análisis

Andrea Merino Mesa
T.I. 1034991011

Ejercicio: La solución está en las siguientes páginas, por orden y facilidad de lectura se hace un salto después de cada justificación.

Paquete objtaller3 – ObjTaller3:

```
package objtaller3;
import compras.*;
```
//Para importar todo el paquete en vez de una sola clase se usa punto asterisco (.*).

```
import gestionHumana.Empleado;
import java.util.ArrayList;
```
// Para poder usar la clase ArrayList más adelante debemos importar el paquete del cuál esta hace parte y especificar que queremos usar dicha clase (java.util.ArrayList).

```
public class ObjTaller3 {

    public static void main(String[] args) {
```
// El String [] hace parte fundamental de la estructura del método main.

```
        Producto p1 = new Producto(1, "Escoba", "Aseo");
        Producto p2 = new Producto(2, "Camisa", "Ropa");
        Producto p3 = new Producto(3, "Trapera", "Aseo");
        Producto p4 = new Producto(4, "Pantalon", "Ropa");
        Producto p5 = new Producto(5, "Jabon", "Aseo");
        Empleado emp1 = new Empleado(405, "Juan", "Ingeniero");
        ArrayList<Producto> productos1 = new ArrayList<>();
        productos1.add(p1);
        productos1.add(p3);
        OrdenCompra orden1 = new OrdenCompra(101, "Aseo", emp1, productos1);
```
// Pues es el valor que el ejercicio nos indica que debemos poner, además sigue una secuencia porque la próxima vez que intentemos crear un nuevo objeto de tipo OrdenCompra se le suman ciento uno (101, 202 ...)

```
        System.out.println(Producto.getTotalProductosPedidos());
        orden1.agregarProducto(p4);
        System.out.println(Producto.getTotalProductosPedidos());
        orden1.agregarProducto(p5);
```

// Porque los métodos que se podrían hacer con p5 son agregar o retirar producto, y como p5 no estaba agregado a ninguna orden, lógicamente lo que se debe hacer es agregarlo, porque retirarlo no tendría sentido.

```
        System.out.println(Producto.getTotalProductosPedidos());
        System.out.println("Orden " + orden1.codigo + " creada");

        Empleado emp2 = new Empleado(128,"Susana", "Administradora de sucursal");
        ArrayList<Producto> productos2 = new ArrayList<>();
        productos2.add(p2);
        productos2.add(p4);
        OrdenCompra orden2 = new OrdenCompra(202, "Ropa", emp2, productos2);
        System.out.println(Producto.getTotalProductosPedidos());
        System.out.println(emp2.cedula + " va a retirar producto");
        orden2.retirarProducto(emp2, p4);
        System.out.println(Producto.getTotalProductosPedidos());
        orden2.retirarProducto(emp1, p2);
```

// El método retirarProducto requiere que se le pase un valor de tipo Empleado, que hasta el momento podría ser emp2 o emp2, pero necesitamos que no se retire ningún producto y para ello quien lo llame no debe tener cargo Administrador, lo que nos deja con emp1.

```
        System.out.println(Producto.getTotalProductosPedidos());
    }
}
```

Paquete compras - OrdenCompra:

```
package compras;
import gestionHumana.Empleado;
import java.util.ArrayList;
```

// Necesitamos importar la clase empleado y para ello debemos hacerlo desde el paquete donde está, gestionHumana.

```
public class OrdenCompra {
```

//El nombre del método constructor debe ser el mismo que el de la clase.

```
   public int codigo;
    private String tipo;
    private Empleado comprador;
    public ArrayList<Producto> productos;
```

//Definimos la visibilidad de los atributos, para cumplir con el encapsulamiento, solo ponemos públicos aquellos que se llaman desde otra clase fuera del paquete y el resto los ponemos privados.

```
public OrdenCompra(int codigo, String tipo, Empleado comprador,
        ArrayList<Producto> productos) {
    this.codigo = codigo;
    this.tipo = tipo;
    this.comprador = comprador;
    this.productos = productos;
    Producto.totalProductosPedidos += productos.size();
  }

  public void agregarProducto(Producto producto) {
    if (producto.tipo.equals(tipo)) {
      productos.add(producto);
      Producto.totalProductosPedidos++;
    }
  }
```

//Lo recomendado es tener los métodos con visibilidad pública.

```
  public void retirarProducto(Empleado empleado, Producto producto) {
    if (!empleado.tengoPermiso()) {
      return;
    }
```

//Lo recomendado es tener los métodos con visibilidad pública. Además, en este método dejamos el return sólo ya que si el empleado no tiene permiso de retirar Productos no podemos hacer nada más con él sino salirnos del método y seguir la ejecución.

```
    retirarProducto(producto);
  }

  private void retirarProducto(Producto producto) {
    for (int i = 0; i < productos.size(); i++) {
      if (producto.getCodigo() == productos.get(i).getCodigo()) {
        productos.remove(i);
        producto.totalProductosPedidos--;
        producto.imprimirNombre();
        System.out.println(" retirado");
        break;
      }
    }
  }
```

//productos.size() no permite medir cuanto mide el ArrayList para así detener el ciclo cuando llegue a esa cantidad de iteraciones.

```
    public void descontar() {
        Producto.totalProductosPedidos -= productos.size();
    }

}
```

// Definimos el método con un void pues no devuelve nada y de no hacerlo mostraría error.

Paquete compras - Producto:

```
package compras;

public class Producto {

  public final int codigo;
  private String nombre;
    String tipo;
   public static int totalProductosPedidos;
```

//Definimos la visibilidad de los atributos, para cumplir con el encapsulamiento, solo ponemos públicos aquellos que se llaman desde otra clase fuera del paquete, dejamos los que se llaman desde otra clase del mismo paquete como están para indicar que tengan visibilidad de paquete y el resto los ponemos privados.

```
    public Producto(int codigo, String nombre, String tipo) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.tipo = tipo;
    }
```

//Lo recomendado es tener los métodos con visibilidad pública.

```
    void imprimirNombre() {
        System.out.print(nombre);
    }

    public void setCodigo(int codigo) {
return;
    }
```

//Como el atributo código es de tipo final, no podemos reasignarle un valor por lo que lo único que queda es salirse del atributo y continuar la ejecución usando return.

```
    public int getCodigo() {
        return codigo;
```

```
    }
//Porque retorna un valor de tipo entero.

    public static int getTotalProductosPedidos() {
        return totalProductosPedidos;
    }
}
//Lo recomendado es tener los métodos con visibilidad pública.
```

Paquete GestionHumana – Empleado:

```
package gestionHumana;
//Porque es el paquete donde la clase Empleado se encuentra.

public class Empleado {

    public final long cedula;
    private String nombre;
    private String cargo;
//Definimos la visibilidad de los atributos, para cumplir con el encapsulamiento, solo ponemos públicos aquellos que se llaman desde otra clase fuera del paquete y el resto los ponemos privados.


    public Empleado(long cedula, String nombre, String cargo) {
        this.cedula = cedula;
        this.nombre = nombre;
        this.cargo = cargo;
    }

   public boolean tengoPermiso() {
        return cargo.contains("Administrador");
    }
}
//Lo recomendado es tener los métodos con visibilidad pública.
```