**Programación ll – Trabajo Práctico Integrador**

**2do. Cuatrimestre 2023**

**Segunda Parte**

**Profesores:** José Nores y Adrián Cáceres

**Comisión:** 03

**Integrantes Grupo 6:**

**Nombre y Apellido**: Joaquin Hernández Durand

**DNI**: 44131754

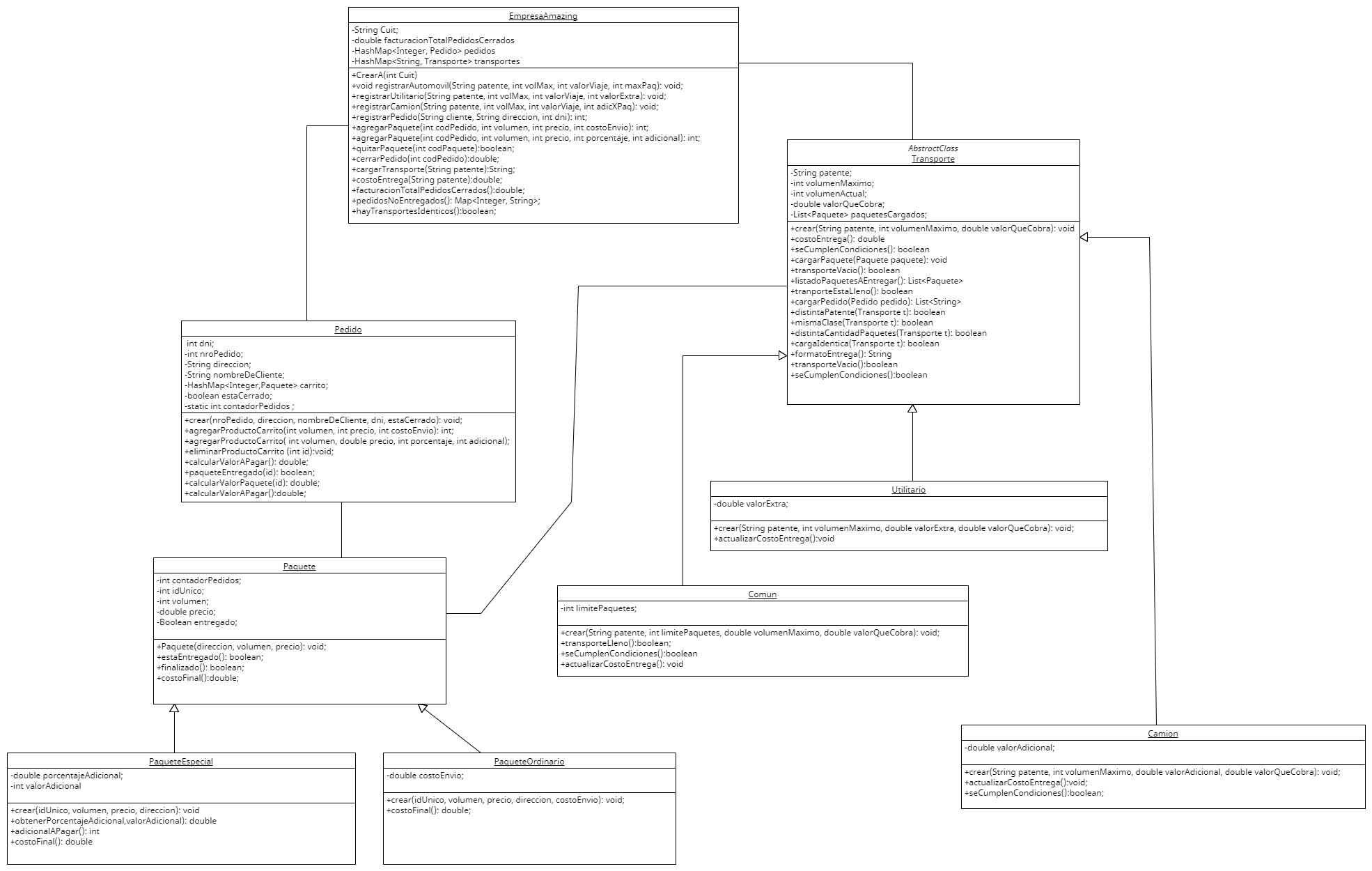
**Mail**: [joacohernandez157@gmail.com](mailto:joacohernandez157@gmail.com)

**Nombre** **y** **Apellido**: Joaquin Flores

**DNI**: 43663411

**Mail**: [joaco5flores@gmail.com](mailto:joaco5flores@gmail.com)

**Fecha de entrega:** 30/10/2023

****

**Introducción**

El propósito de este informe es presentar el diseño de un sistema que simula el proceso de entrega de paquetes para la gigante empresa de comercio electrónico, Amazing. Este sistema permitirá gestionar los pedidos de los clientes, calcular el costo total de envío y distribuir los paquetes a través de una flota de transportes. La empresa, como parte de su operativa, recibe los pedidos y administra el reparto mediante una variedad de vehículos de transporte.

**Especificaciones y Funcionalidades**

**Aplicación de conceptos vistos:**

**Polimorfismo:**

En este programa el polimorfismo para la interfaz que toma la forma de EmpresaAmazing.

**Herencia:**

En este programa, se utilizó la herencia para las clases "Pedido Especial" y "Pedido Ordinario," las cuales heredan de la clase "Pedido." También se empleó para las clases "Transporte Utilitario," "Transporte Común," y "Camión," que heredan de la clase "Transporte

**Abstracción:**

La clase Transporte es una clase abstracta para modelar las subclases “Común”, “Utilitario“ y “Camión” las cuales heredan sus atributos y métodos. A demás, “Transporte” tiene un método actualizarCostoEntrega() el cual es abstracto y se sobre escribe en sus subclases.

**Sobre escritura:**

En la implementación, se utilizó el método costoFinal(), que según si era un paquete tipo especial u ordinario lo calculaba de manera diferente.

En cada subclase de “Transporte” se sobrescribió el método seCumplenCondiciones() de acuerdo a los tipos de paquete que acepta cada tipo de transporte. El método actualizarCostoEntrega() se sobrescribió dependiendo del tipo de transporte, este tomara o no en cuenta un valor adicional. Por último, la clase “Común” también sobrescribió el método transporteLleno() ya que esta subclase cuenta con un límite de paquetes.

**Sobrecarga:**

En la implementación, se lo utilizo para el método agregarPaquete(), que, según los parámetros pasados, detectaba si era un paquete tipo ordinario o especial.

**Registro de Pedidos**

Cada pedido es identificado por un número único y registra los detalles del cliente, como el nombre, la dirección y los paquetes solicitados.

Cada paquete, considerado como un elemento individual del carrito de compras, tiene su identificador único, volumen y precio.

**Paquetes**

Existen dos tipos de paquetes: ordinarios y especiales.

Los paquetes ordinarios incluyen un costo de envío adicional al precio base.

Los paquetes especiales, además del envío rápido, tienen costos adicionales basados en su volumen. Un adicional se agrega si el volumen es superior a 3000 y se duplica si supera los 5000.

**Entregas** **de** **Paquetes**

Cada paquete se entrega individualmente, no necesariamente junto con todo el pedido. La dirección de entrega corresponde a la dirección del cliente.

Los pedidos pueden incluir tanto paquetes ordinarios como especiales.

**Flota** **de** **Transporte**

La flota de transportes está compuesta por diferentes vehículos: automóviles, utilitarios y camiones.

Cada tipo de transporte tiene restricciones en cuanto al tipo de paquete que puede transportar y su capacidad máxima.

**Asignación** **y** **Carga** **de** **Transporte**

La empresa asigna un vehículo de transporte para las entregas.

Los paquetes especiales se cargan primero, especialmente en utilitarios y camiones, y luego los paquetes ordinarios si hay capacidad disponible en ese transporte.

En respuesta a la solicitud de Amazing, la gigante empresa de comercio electrónico, se ha diseñado un sistema de entrega de paquetes para gestionar los pedidos de los clientes y administrar el transporte de manera eficiente. A continuación, se presenta el diseño detallado del sistema, considerando los distintos tipos de paquetes, vehículos de transporte y las operaciones involucradas en el proceso.

**Añadidos para re-entrega**

**•**Se encapsularon la mayor cantidad de métodos posibles para facilitar la lectura del código y evitar errores en la implementación.

**•**Con tal de sacar el mayor provecho a los atributos del POO, se hizo abstracta la clase transporte. Para que sus hijos saquen el mayor provecho.

**•**Al método transportes idénticos se agregó un acumulador booleano, ya que no detectaba si TODOS los paquetes eran iguales, solo detectaba uno.

**•**Al método removerPaquete, se lo encapsulo y se ejecuta directamente de la clase pedido.

**•**Transporte ahora es una clase abstracta.

**EmpresaAmazing**

**IREP:**

* Los valores del diccionario pedidos tienen que ser diferente de null.
* Los valores del diccionario transportes tienen que ser diferente de null.
* facturacionTotalPedidosCerrados tiene que ser > 0.
* cuit tiene que ser diferente de null.

**PaqueteEspecial**

**IREP:**

* idUnico deber ser un entero no repetido.
* Volumen debe ser un numero entero mayor que 0.
* Dirección debe ser diferente a null.
* Precio debe ser mayor a 0.
* porcentajeAdicional debe ser un double mayor que 0.
* valorAdicional debe ser un entero mayor que 0.

**PaqueteOrdinario**

**Datos:**

* int idUnico;
* int volumen;
* double precio;
* String direccion;
* boolean entregado;
* double costoEnvio;

**IREP:**

* idUnico deber ser un entero no repetido.
* volumen debe ser un entero positivo.
* Precio debe ser un entero mayor o igual a 0.
* Dirección debe ser diferente a null.
* costoEnvio debe ser un double mayor o igual que 0.

**TransporteComun**

**IREP:**

* Patente debe ser una combinación de enteros y números no nulos.
* volumenMaximo, y limitesPaquetes deben ser enteros positivos.
* valorqueCobra debe ser un double mayor que 0.
* volumenActual no puede superar volumenMaximo.

**TransporteUtilitario**

**IREP:**

* Patente debe ser una combinación de enteros y números no nulos.
* volumenMaximo debe ser un entero positivo.
* valorqueCobra y valorExtra deben ser double mayor que 0.
* volumenActual no puede superar volumenMaximo.

**TransporteCamion**

**IREP:**

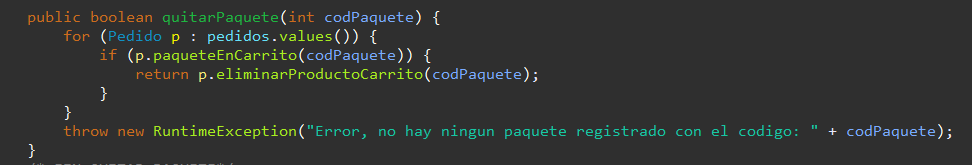
* Patente debe ser una combinación de enteros y números no nulos.
* volumenMaximo debe ser un entero positivo.
* valorqueCobra y valorAdicional deben ser double mayor que 0.
* volumenActual no puede superar volumenMaximo.

**Pedido**

**IREP:**

* nroPedido debe ser un entero único y mayor que 0.
* dirección debe ser un String no null y valido.
* nombreCliente debe ser un String valido.
* dni debe ser un numero diferente a null y con una longitud acorde al dni del país
* contadorPedidos debe ser mayor a 0
* carrito no puede tener valores null

**Análisis de complejidad**

****

public boolean quitarPaquete(int codPaquete) {

for (Pedido p : pedidos.values()) { (    🡪 O(n)

if (p.paqueteEnCarrito (p, codPaquete)) {  🡪 O(G(n))\*O(n)

return p.eliminarProductoCarrito(codPaquete); 🡪O(T(n))

}

}

throw new RuntimeException("Error, no hay ningun paquete registrado con el codigo: " + codPaquete);

}

K(n) = O(n + 1n + 1)

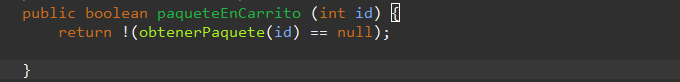
Regla2 = O(n) + O(1n) + O(1)

Regla3 = O(n) + O(1) \* O(n) + O(1)

Regla3= O(n) + O(1)

Regla2= O(Max{n,1})

Regla1= O(n)

K(n) = O(n)

public boolean paqueteEnCarrito(int id){

return !(obtenerPaquete(id) == null); 🡪 O(1)

}

G(n) = O(1)

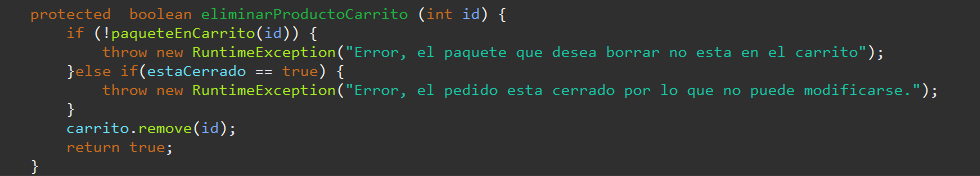


public Paquete obtenerPaquete(int id){

return carrito.get(id); 🡪 O(1)

}

F(n) = O(1)



Protected Boolean eliminarProductoCarrito(int id){

If (!paqueteEnCarrito(id)){ 🡪 O(G(n))

throw new Runtimexception(“Error el paquete que desea borrar no esta en el carrito.”);

}else if(estaCerrado == true){ 🡪 O(1)

throw new RuntimeException(“Error, el pedido esta cerrado por lo que no puede modificarse.”);

}

carrito.remove(id); 🡪 O(1)

return true; 🡪 O(1)

}

T(n)= O(1)