**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:**

El polimorfismo se refiere a la capacidad de diferentes objetos de una jerarquía de clases para responder de manera diferente a una misma solicitud o mensaje. En otras palabras, objetos de diferentes clases pueden implementar la misma interfaz o método de manera específica para sus propias necesidades.

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

**Herencia:**

La herencia en la programación orientada a objetos (POO) es un concepto que permite que una clase (llamada clase derivada o subclase) herede propiedades y métodos de otra clase (llamada clase base o superclase). La subclase puede heredar y extender las características de la superclase, lo que facilita la reutilización de código y la creación de una jerarquía de clases.

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

**Sobre escritura:**

La sobre escritura es un concepto que permite a una subclase proporcionar una implementación específica para un método que ya existe en su superclase. Esto significa que la subclase puede reemplazar o "sobrescribir" la implementación del método heredado, adaptándolo a sus propias necesidades.

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

**Sobrecarga:**

La sobrecarga en programación orientada a objetos (POO) se refiere a la capacidad de definir múltiples métodos con el mismo nombre en una clase, pero con diferentes parámetros. Estos métodos comparten el mismo nombre, pero se diferencian por el número o tipo de parámetros que aceptan.

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.

**TAD EmpresaAmazing**

**Datos:**

* HashMap<Integer, Pedido> pedidos;
* HashMap <String, Transporte>transportes;
* Double facturacionTotalPedidosCerrados;
* String cuit;

**Operaciones:**

* crear(String cuit): void;
* registrarAutomovil(String patente, int volMax, int valorViaje, int maxPaq): void;
* registrarUtilitario(String patente, int volMax, int valorViaje, int valorExtra): void;
* registrarCamion(String patente, int volMax, int valorViaje, int adicXPaq): void;
* registrarPedido(String cliente, String direccion, int dni) : int;
* agregarPaquete(int codPedido, int volumen, int precio, int costoEnvio): int;
* agregarPaquete(int codPedido, int volumen, int precio, int porcentaje, int adicional): int;
* quitarPaquete(int codPaquete): boolean;
* cerrarPedido(int codPedido): double;
* cargarTransporte(String patente): String;
* costoEntrega(String patente): double;
* facturacionTotalPedidosCerrados(): double;
* Map<Integer, String> pedidosNoEntregados(): Map<Integer, String>;
* hayTransportesIdenticos(): boolean;
* toString(): String;

**IREP:**

* registrarAutomovil, registrarUtilitario y registrar Camión, toman una patente, si esa patente ya está en el sistema genera una excepción.
* agregarPaquete genera una excepción si el pedido no esta registrado o si el pedido ya está finalizado.
* quitarPaquete genera una excepción si no encuentra el paquete en el pedido.
* cerrarPedido genera una excepción si no existe el pedido o si el pedido ya fue cerrado anteriormente.
* cargarTransporte genera una excepción si no encuentra el transporte con la patente dada o si el transporte está lleno.
* costoEntrega genera una excepción si no existe el transporte con la patente dada o si el transporte está vacío.

**TAD PaqueteEspecial**

**Datos:**

* int idUnico;
* int volumen;
* double precio;
* boolean entregado;
* String direccion;
* double porcentajeAdicional;
* int valorAdicional;

**Operaciones:**

* crear(int volumen, double precio, String direccion): void;
* fueEntregado(): boolean;
* paqueteEntregado(): boolean;
* adicionalAPagar (): int;
* costoFinal(): double;
* toString(): String;

**IREP:**

* idUnico deber ser un entero no repetido.
* Volumen debe ser un numero entero mayor que 0.
* porcentajeAdicional debe ser un double mayor que 0.
* valorAdicional debe ser un entero mayor que 0.

**TAD PaqueteOrdinario**

**Datos:**

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

**Operaciones:**

* crear(int volumen, double precio, String direccion, double costoEnvio): void;
* paqueteEntregado(): void;
* fueEntregado(): boolean;
* costoFinal(): double;
* toString(): String;

**IREP:**

* idUnico deber ser un entero no repetido.
* volumen debe ser un entero positivo.
* costoEnvio debe ser un double mayor o igual que 0.

**TAD TransporteComun**

**Datos:**

* String patente;
* Int volumenMaximo;
* Int volumenActual;
* double valorQueCobra;
* Int limitePaquetes;
* List<Paquete> paquetesCargados;

**Operaciones:**

* crear(String patente, int volumenMaximo, int limitePaquetes, double valorQueCobra): void;
* aumentarVolumen(int volumenPaquete): void;
* costoEntrega(): double;
* seCumplenCondiciones(Paquete p): boolean;
* cargarPaquete(Paquete paquete): void;
* transporteVacio(): boolean;
* List<Paquete> listadoPaquetesAEntregar: List<Paquete>;
* superaLimite(): boolean;
* transporteLLeno(): boolean;
* transporteEstaLLeno(): void;
* toString(): String;

**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.
* aumentarVolumen genera excepción si recibe un número menor que 0.
* transporteEstaLLeno genera excepción si el transporte esta lleno.

**TAD TransporteUtilitario**

**Datos:**

* String patente;
* Int volumenMaximo;
* Int volumenActual;
* double valorQueCobra;
* double valorExtra;
* List<Paquete> paquetesCargados;

**Operaciones:**

* crear(String patente, int volumenMaximo, double valorExtra, double valorQueCobra): void;
* aumentarVolumen(int volumenPaquete): void;
* costoEntrega(): double;
* seCumplenCondiciones(Paquete p): boolean;
* cargarPaquete(Paquete paquete): void;
* transporteVacio(): boolean;
* List<Paquete> listadoPaquetesAEntregar: List<Paquete>;
* superaLimite(): boolean;
* transporteEstaLLeno(): void;
* calcularCostoViaje(): void;
* toString(): String;

**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.
* aumentarVolumen genera excepción si recibe un número menor que 0.
* transporteEstaLLeno genera excepción si el transporte está lleno.

**TAD TransporteCamion**

**Datos:**

* String patente;
* Int volumenMaximo;
* Int volumenActual;
* double valorQueCobra;
* double valorAdicional;
* List<Paquete> paquetesCargados;

**Operaciones:**

* crear(String patente, int volumenMaximo, double valorAdicional, double valorQueCobra): void;
* aumentarVolumen(int volumenPaquete): void;
* costoEntrega(): double;
* seCumplenCondiciones(Paquete p): boolean;
* cargarPaquete(Paquete paquete): void;
* transporteVacio(): boolean;
* List<Paquete> listadoPaquetesAEntregar: List<Paquete>;
* superaLimite(): boolean;
* transporteEstaLLeno(): void;
* calcularCostoViaje(): void;
* toString(): String;

**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.
* aumentarVolumen genera excepción si recibe un número menor que 0.
* transporteEstaLLeno genera excepción si el transporte está lleno.

**TAD Pedido**

**Datos:**

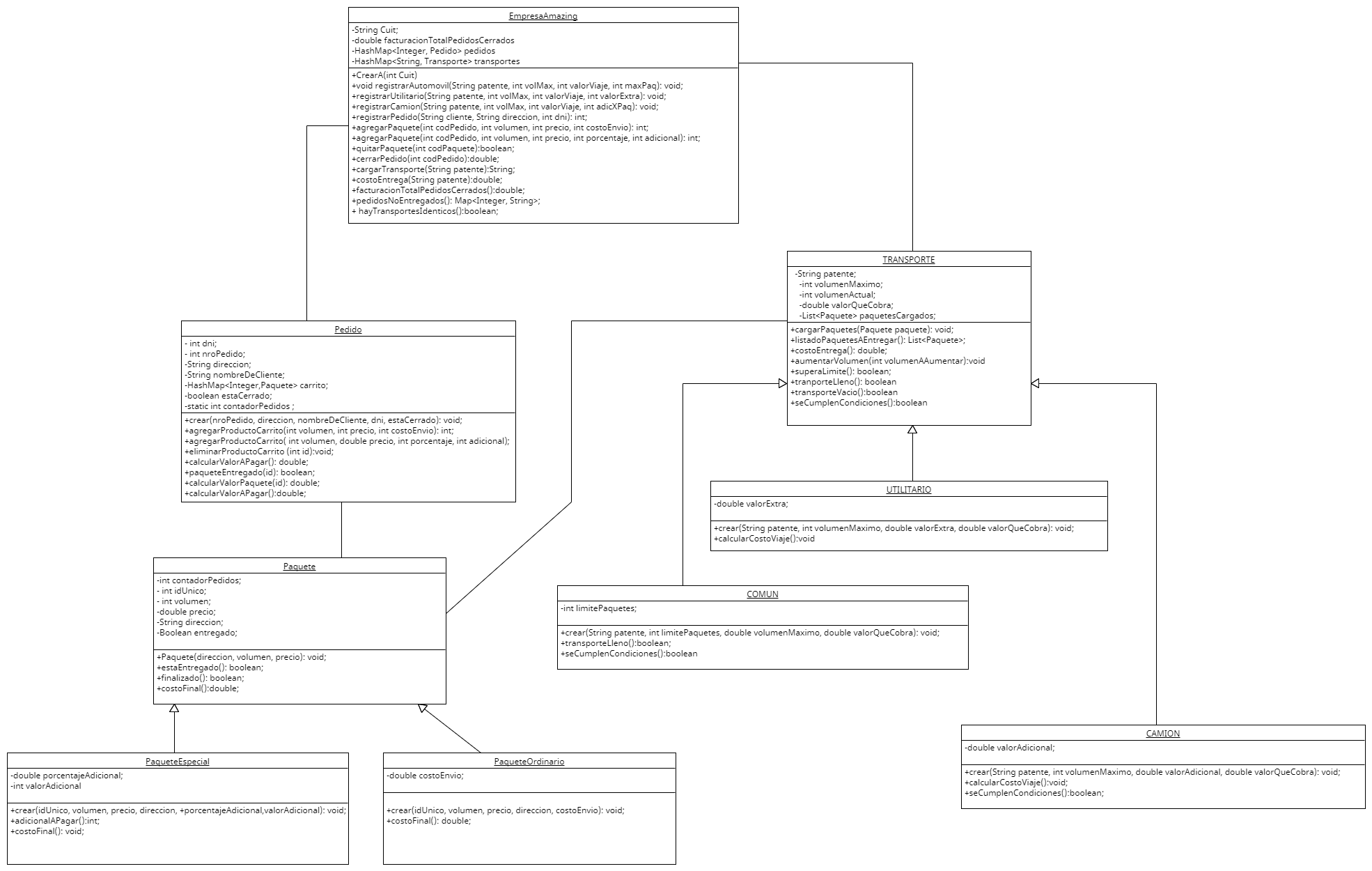
* int nroPedido;
* String dirección;
* String nombreDeCliente;
* Int dni;
* boolean estCerrado;
* int contadorPedidos;
* HashMap <Integer,Paquete> carrito;

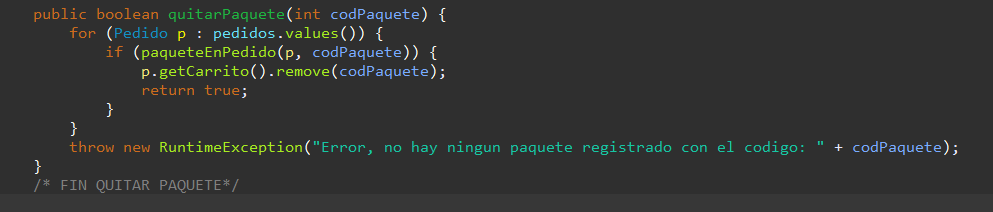
**Operaciones:**

* crear(int nroPedido, int dni, String dirección, String nombreDeCliente, boolean estaCerrado): void;
* agregarProductoCarrito(int volumen, double precio, int costoEnvio): void;
* agregarProductoOrdinario(Integer id, PaqueteOrdinario p): void;
* agregarProductoCarrito(int volumen, double precio, int porcentaje, int adicional): void;
* agregarProductoEspecial(Integer id, PaqueteEspecial p): void;
* eliminarProductoCarrito(int id): void;
* calcularValorAPagar(): double;
* paqueteEnCarrito (int id): boolean;
* paqueteEntregado(int id): boolean;
* calcularValorPaquete(int id): double;
* obtenerPaquete(int id): Paquete;
* hayNumerosEnString(String s): boolean;
* finalizarPedido(): void;
* toString(): String;

**IREP:**

* nroPedido debe ser un entero único y mayor que 0.
* direccionCliente debe ser un String no null y valido.
* nombreCliente debe ser un String valido.
* agregarProductoOrdinario y agregarProductoEspecial generan excepción si ya existe un paquete con esa id o si el pedido cerro.
* eliminarProductoCarrito genera excepción si el paquete no esta en el carrito o si el pedido está cerrado.



**Análisis de complejidad**

public boolean quitarPaquete(int codPaquete) {

for (Pedido p : pedidos.values()) { (   —-----> 2n

if (paqueteEnPedido(p, codPaquete)) {  —-- >n\* (7)

p.getCarrito().remove(codPaquete);  —> 1

return true; —--- > 1

}

}

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

}

Complejidad quitarPaquete (int codPaquete), tendrá un orden lineal O(n). Por lo tanto, la suma es F(N) = 9n+2.

El álgebra de Ordenes sería el siguiente:

O(N) = O(9n+2)

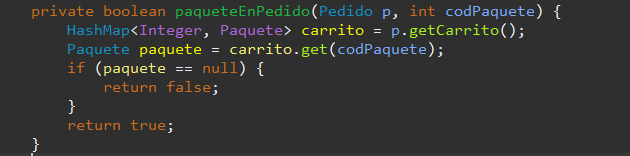
  =  Regla 2 —>O(9n) + O(2)

= Regla 3  —> O(9) \* O(n) + O(2)

= Regla 4  —>O(1) \* O(n) + O(1)

= Regla 3  —> O(n) + O(1)

=Regla 2 —> O(Max {n,1})

= Regla 1  —>O(n)

private boolean paqueteEnPedido(Pedido p, int codPaquete) {

HashMap<Integer, Paquete> carrito = p.getCarrito(); —--> 2

Paquete paquete = carrito.get(codPaquete); —-> 2

if (paquete == null) {   —--> 2

return false; —-> 1

}

return true; —->1

}

Por lo tanto PaqueteEnPedido tiene un orden de complejidad constante O(1).