*****EMPRESA AMAZING*****

Datos:

String cuit
String nombre
Transporte: HashMap <String, Transporte> transportesMap
Pedido: HashMap <Integer, Pedido> pedidos hashMap()
Double totalPedidosCerrados

Operaciones:

+crear(String cuit):
+int registrarPedido(String nombre, String domicilio, double dni) : devuelve un numero de pedido unico
+int agregarPaquete(int numPedido, int volume, double precio, double costoEnvio): en el caso de los paquetes ordinarios, devuelve un numero de paquete único, además genera error si el pedido no existe o si el pedido se encuentra cerrado
+int agregarPaquete(int numPedido, int volumen, double precio, int porcentaje, double adicional): en el caso de los paquetes especiales, devuelve un numero de paquete único, además genera error si el pedido no existe o si el pedido se encuentra cerrado
+ boolean quitarPaquete(int codPaquete) : recibe un código de paquete y devuelve verdadero si se pudo eliminar el paquete del pedido asociado. Genera error cuando se quiere eliminar un paquete de un pedido que no existe o este cerrado.
+ double cerrarPedido(int codPedido): recibe un numero de pedido, cierra el pedido y devuelve el valor total de ese pedido. Genera error si se quiere cerrar un pedido que no existe o ya este cerrado previamente.
+double obtenerImportePorDni(dni): recibe un dni y devuelve el monto a abonar de todos sus pedidos. Genera error si el cliente no esta registrado.
+ double facturacionTotalPedidosCerrados(): devuelve el valor total de los pedidos cerrados únicamente.
+ Map<Integer,String> pedidosNoEntregados(): devuelve un listado con los pedidos cerrados que no fueron entregados en su totalidad.
+ void registrarUtilitario(): devuelve error si el utilitario ya esta registrado.
+void registrarCamion(): devuelve error si el camion ya esta registrado.
+void registrarAutomovil(): devuelve error si el automovil ya esta registrado.
+ boolean patenteYaRegistrada(): valor lógico
+arrayList<Paquete> paquetesCerrados(): devuelve un arrayList de paquetes cerrados.
+ String cargarTransporte(): devuelve un StringBuilder de los paquetes cargados, da error si la patente no esta cargada en sistema, si no hay paquetes finalizados arroja un aviso.
+ double costoEntrega(): devuelve un double con precio, da error si no hay transporte para verificar o si la patente no existe.
+ boolean hayTransportesIdenticos(): devuelve verdadero si hay al menos dos transportes idénticos, genera error si no hay al menos 2 transportes para comparar.

Metodos internos que nos sirven para los métodos principales:

- Pedido obtenerPedido(int numeroPedido): Devuelve el pedido al que pertenece su número.
- Cliente buscarCliente(int dni): devuelve un cliente
- Transporte obtenerTransportePorPatente(String patente): Devuelve el transporte al que pertenece la patente. Arroja error si no hay transporte con dicha patente.
- List<Transporte> obtenerListaDeTransportes() : devuelve lista de transportes de la empresa

****Pedidos****

Datos:
int numPedido
Cliente cliente
Map<Integer,Paquete> paquetesMap
double valor
boolean pedidoCerrado

Operaciones:
-crear(String nombre, String dirección, int dni)
-boolean esPedidoCerrado(): retorna el estado del pedido actual.
-int lenPaquetes(): devuelve la cantidad de paquetes del pedido
-ArrayList<Paquete> listaPaquetesPorPedido(): se obtiene la lista de paquetes del pedido.
-double cerrarPedido(): cambia el estado de pedidoCerrado y devuelve el valor del mismo. Genera error si ese pedido ya estaba cerrado anteriormente.
-void agregarPaquete(int numPaquete, int volumen, double precio, double costoEnvio) para los paquetes ordinarios
-void agregarPaquete(int numPaquete, int volumen, double precio, int porcentaje, double adicional) para los paquetes especiales.
-void quitarPaquete(int codPaquete): remueve paquete del pedido.
boolean tienePaquete (): verifica si el pedido tiene el paquete que se le pasa por parámetro.
-double valorTotalDeEstePedido(): retorna el valor del pedido
-boolean tienePaquetesParaEntregar(): Valor lógico, revisa si algún paquete del listado todavía no fue entregado.
-String nombreCliente(): se obtiene el nombre del cliente que hizo el pedido

****Cliente****

Datos:

-int dni

-String nombreCompleto

-String direccion

Operaciones:

- crearCliente(String nombreCompleto, String dirección, int dni).

****Paquete** (Clase abstracta)**

Datos:

-int numPaquete

-int volumen

-double precio

-boolean entregado

-int numPedido

-String dirección

-String clase

Operaciones:

-crear(int numPaquete, int volumen, double precio)
abstract double calcularPrecioPorEstePaquete(): devuelve un precio, se calcula en las sub-clase.

-abstract boolean sonAtributosIguales(Paquete otroPaquete): valor lógico que se compara en las sub-clase.
-boolean isEntregado(): valor lógico para saber si se entregó el paquete.

Métodos para verificar si son paquetes iguales:
-boolean sonPaquetesIdenticos(Paquete otroPaquete): Valor lógico, recibe un paquete y lo compara consigo mismo, verificando si tiene mismo precio,volumen, clase y atributos
-boolean mismoVolumen(Paquete otroPaquete)
-boolean mismaClase(Paquete otroPaquete)

-boolean mismoPrecio(Paquete otroPaquete)

****PaqueteOrdinario** (hereda de Paquete)**

Datos:

- double costoEnvio

Operaciones:
-crear(int numPaquete, int volumen, double precio, double costoEnvio)
- double calcularPrecioPorEstePaquete(): sobreescribe el método y devuelve el valor del paquete + el costo de envio
-boolean sonAtributosIguales(Paquete otroPaquete): sobrescribe el método sonAtributos iguales, comparando si este paquete tiene el mismo costo de envió que el otro paquete que se le pasa por parámetro.

****PaqueteEspecial** (hereda de Paquete)**

Datos:

-int porcentajeEntregaRapida

- double valorAdicional

Operaciones:
-crear(int numPaquete, int volumen, double precio, int porcentajeEntregaRapida, double valorAdicional)
- double calcularPrecioPorEstePaquete(): sobrescribe el método y devuelve el valor del paquete + el porcentaje deentrega rápida y/o valor adicional
- boolean sonAtributosIguales(Paquete otroPaquete): sobrescribe el método y compara si el paquete tiene el mismo porcentajeEntregaRapida y el mismo valorAdicional que el paquete que se le pasa por parámetro.

****Transporte** Clase abstracta**

Datos:
-String patente

-Double volumen máximo de carga

-Double precio base viaje

-String tipo

-Paquete : ArrayList paquetesEnTransporte

Operaciones:

-crear():

-boolean sonTransportesIdenticos(Transporte otro): valor lógico que verifica si el transporte tiene distinta patente, mismo tipo y carga que el transporte que recibe por parámetro.

-boolean distintaPatente(Transporte otroTransporte): valor lógico que compara la patente del transporte que lo llama con la que recibe por parámetro.

-mismoTipo(Transporte otroTransporte): valor lógico que compara el tipo de transporte del mismo que lo llama con el que recibe por parámetro.

-boolean mismaCarga(Transporte otro): método que verifica si los paquetes del transporte son iguales al transporte pasado por parámetro, verificando que sean iguales, usando el método sonPaquetesIdenticos(Paquete otroPaquete).

-abstract double calcularPrecioPorEsteTransporte (): método abstracto,
busca el valor en los hijos y devuelve precio

-abstract String cargarEsteTransporte(ArrayList<Paquete> paquetes):
método abstracto que carga los paquetes en el transporte

****Automovil** (Hereda de Transporte)**

Datos:

-Int limite máximo de paquetes

Operaciones:

-crear(String patente, double volumenMaximoCarga, double
precioBase, intlimiteMaxPaquetes)

-calcularPrecioPorEsteTransporte(): sobrescribe el método devolviendo el valor del transporte.

- String cargarEsteTransporte(ArrayList<Paquete> paquetes): recibe una lista de paquetes y devuelve un string con los paquetes que puede cargar automóvil: Paquetes Ordinarios.

****Utilitario** (Hereda de Transporte)**

Datos:

-Double valorExtraPorViaje

-Double precioBase

Operaciones:

-crear(String patente, double volumenMaximoCarga, double precioBase, double valorExtraPorViaje)

- double calcularPrecioPorEsteTransporte(): sobrescribe el método y calcula el precio sumándole al precio base (en el caso que cargue >= 3 paquetes) el valorExtraPorViaje

-String cargarEsteTransporte(ArrayList<Paquete> paquetes): recibe un array de paquetes y devuelve un string con los paquetes que pudo cargar en el utilitario, cargando primero los paquetes especiales y luego los ordinarios

****Camion** (Hereda de Transporte)**

Datos:

-double valorAdicionalPorPaquete

Operaciones:

-crear(String patente, double volumenMaximoCarga, double precioBase, double valorAdicionalPorPaquete)

-double calcularPrecioPorEsteTransporte(): sobrescribe le método y devuelve el valor del transporte sumándole al precio base, la cantidad de paquetes multiplicada por el valorAdicionalPorPaquete.

-String cargarEsteTransporte(ArrayList<Paquete> paquetes): recibe un array de paquetes y devuelve un string con los paquetes que pudo cargar, siendo estos solo paquetes especiales que tengan un volumen mayor a 2000.

USO DE TECNOLOGÍAS JAVA

Para tener un control a la hora de gestionar los pedidos en Amazing, nos vimos en la necesidad de contar con alguna herramienta dinámica que nos permitiera agregar, quitar, consultar “x” cosa, ya que no podríamos saber la cantidad de elementos que íbamos a tener. Por lo tanto, utilizamos la herramienta Hashmap, con clave = numPedio y valor = Pedido, que, a su vez, esos pedidos también poseen su hashmap de

paquetes<codPaquete, Paquete>. En el caso de los Transportes, fue más cómodo acceder a ellos utilizando otro HashMap <String patente, Transporte>, donde cada transporte tenia su array de paquetes.

En el caso de las Clases Paquete y Transporte, se decidió que fueran abstractas ya que en ambos casos había diferentes tipos, pero que compartían datos y operaciones.

Como la idea de esta empresa es poder agregar más transportes y tipos de paquetes a futuro, lo más conveniente fue unificar los datos que compartían y que cada subclase operara a su manera el calcular el costo y la forma de cargar los paquetes (en el caso de los transportes).

Ya que a lo largo del TP se trabajó con colecciones de Hashmap, utilizando las clave y valores para acceder a ellos, nos vimos en la necesidad de utilizar la herramienta de Stringbuilder, para poder tener una representación más clara del problema planteado, de esa manera se pudo chequear el estado de un pedido, ver la cantidad de paquetes, etc. Fue una herramienta muy útil no solo para poder mostrar al usuario, sino para nosotras como grupo para poder desarrollar el código.

```
public String toString() {
    StringBuilder st = new StringBuilder();
    st.append("numero de pedido >>");
    st.append(numPedido);
    st.append("<< \n");
    st.append(cliente.toString());
    st.append("Pedido cerrado: ");
    st.append(pedidoCerrado);
    st.append("\n");
    st.append("valor total del pedido: $");
    st.append(valor);
    st.append("\nPaquetes: \n");
    for (Paquete paquete : paquetesMap.values()) {

        st.append(paquete);
    }
    st.append("\n");
    return st.toString();
```

También, fue de gran ayuda utilizar un Iterator cuando se nos presentó el problema de “quitarPaquete(int codPaquete)”, ya que al recorrer el arreglo con un foreach, a la hora de eliminar el paquete el sistema nos generaba error. De esta manera pudimos recorrer y remover el elemento, mostrando además, el orden de complejidad que representa este método:

Orden de complejidad

*Método quitarPaquete(int codPaquete);

```
public boolean quitarPaquete(int codPaquete) {
        for (Pedido pedido : pedidosMap.values()) {  --------------(3n+1)
            if (pedido!= null && !pedido.esPedidoCerrado() &&
pedido.tienePaquete(codPaquete)) {-------(n*(3+ 1 +1)
                pedido.quitarPaquete(codPaquete);---  (n* (1+9n+1)
                return true;--    (n*1)
            }
        }
```

```
        throw new RuntimeException("No se encontró el paquete en ningún pedido");
---  1
    }
```

```
    public boolean esPedidoCerrado() {
        return pedidoCerrado;              ------  (1+1)
    }
```

```
public void quitarPaquete(int codPaquete) {
        Iterator<Paquete> iterador = paquetesMap.values().iterator();--(1)
        while (iterador.hasNext()) {  ---------------------------------(n*1)
            Paquete paquete = iterador.next();  ------------------(n*(1))
            if (paquete.getNumPaquete() == codPaquete) -(n*(3))
{                this.valor -= paquete.getPrecio();- (n*(3))
                iterador.remove();  -------(n*(1))
            }
        }
    }
```

(c.aux)= quitarPaquete: 1+n+n+3n+3n+n= 9n+1

(c.aux)=esPedidoCerrado= 2

Respuesta:

quitarPaquete: 3n+1+5n+9n^2+2n+n = $9n^2$+ 11n+1

Orden de complejidad O(n^2)

Justificación con Algebra de Ordenes:

Encerramos a la función en una O grande = O **($9n^2$+ 11n+1)**

Por regla 1 → O(9n^2) + O(11n) + O(1)

Por regla 3 → O(9)*O(n^2)+ O(11)* O(n) + O(1) Por regla 4 → O(1) * O(n^2) + O(1) *O(n) +O(1)
Por regla 3 → O(n^2) + O(n) + O(1)

Por regla 2 → O (max {n^2; n; 1}) Por regla 1 → O(n^2)

JUSTIFICACIÓN DE LOS METÓDOS EN O(1)

Se nos presentó la siguiente consigna:

-9. Dado un transporte cargado devolver el costo que debe pagar la empresa por ese viaje.

```
public double costoEntrega(String patente) {
    if (transportesMap.size()<0) {
        throw new RuntimeException("No hay transporte para verificar");
    }
    Transporte transporte = obtenerTransportePorPatente(patente);
    return transporte.calcularPrecioPorEsteTransporte();
}
```

Este método nos garantiza que va a ser constante ya que, para obtener un transporte, la función obtenerTransportePorPatente(String patente) también accede a los transportes de manera constante, gracias a la simplicidad que nos brinda utilizar un hashMap de transportes.

```
public Transporte obtenerTransportePorPatente(String patente) {
    if (!transportesMap.containsKey(patente)) {
        throw new RuntimeException("No se encontró el transporte con la patente: " + patente);
    }
    return transportesMap.get(patente);

}
```

Para la otra consigna:

-10. Calcular el total de facturación de los pedidos registrados y terminados

Se creo una variable de instancia que inicializara en 0, y que cuando se cierre un pedido a la misma se le sume el valor de ese pedido, por lo tanto el método facturacionTotalPedidosCerrados() solo retorna el valor de la variable, garantizando que sea siempre en O(1) constante.

```
public double facturacionTotalPedidosCerrados() {
    return this.totalPedidosCerrados;

}
```

IREP PARA CADA CLASE

EmpresaAmazing:

- No puede haber dos pedidos con el mismo numPedido.
- La facturación total (totalPedidosCerrados) tiene que ser >0.
- No debe haber dos transportes con la misma patente.
- Para cada paquete cargado en un transporte, debe pertenecer a un pedido registrado en el sistema
- El cuit de la empresa debe ser único
- Para agregar un paquete este no puede pertencer a un pedido cerrado
- No se pueden quitar paquetes de pedidos cerrados
- Un paquete se considera entregado si y solo si pertenece a un pedido cerrado y se cargó en algún transporte
- El costo de entrega de un transporte debe ser > 0
- No se puede cerrar un pedido si el estado de la variable esPedidoCerrado en ese momento es true
- Para verificar si hay transportes idénticos el tamaño del hashmap debe ser >=2
- Al momento de cargar un transporte debe estar registrada la patente y los paquetes no deben estar en estado entregado.
- Para obtener el importe de un cliente el dni debe estar registrado.

Pedido:

- El número de pedido (numPedido) debe ser único y >0.
- El estado del pedido y el estado de sus paquetes deben iniciar en false.
- El valor total del pedido no puede ser negativo.
- Los paquetes asociados al pedido deben ser únicos y no nulos.
- Solo se pueden cerrar pedidos en los que pedidoCerrado sea false.

Cliente:

- El DNI (dni) del cliente debe ser único, no negativo y >0.
- El nombre completo y la dirección del cliente deben tener el length >0.

Paquete:

- El número de paquete (numPaquete) debe ser único y no negativo.
- El volumen y precio del paquete no pueden ser negativos.
- El estado de entrega del paquete debe inicializar en false;
- La dirección debe ser la misma que la dirección del cliente del pedido asociado

PaqueteOrdinario:

- El costo de envío (costoEnvio) no puede ser negativo.

PaqueteEspecial:

- porcentajeEntregaRapida >0
- valorAdicional >0

Transporte:

- Patente única con un lenght >=4
- volumenMaximoCarga >0
- precio base >0
- cada paquete de transporte debe ser diferente y pertencer a un pedido existente.

Automovil:

- limiteMaxPaquetes >0
- los paquetes de automóvil deben tener un volumen <2000 y ser paquetes ordinarios
- paquetesEnTransporte.size() <= limiteMaxPaquetes

Utilitario:

- valorExtraPorViaje >0
- precioBase >0
- se incrementa valorExtraPorViaje si y solo si paquetesEnTransporte.size() >=3

Camion:

- valorAdicionalPorPaquete >0
- Para cada paquete del transporte deben ser paquetes especiales con un volumen >2000

DIAGRAMA DE CLASE