Trabajo practico programación

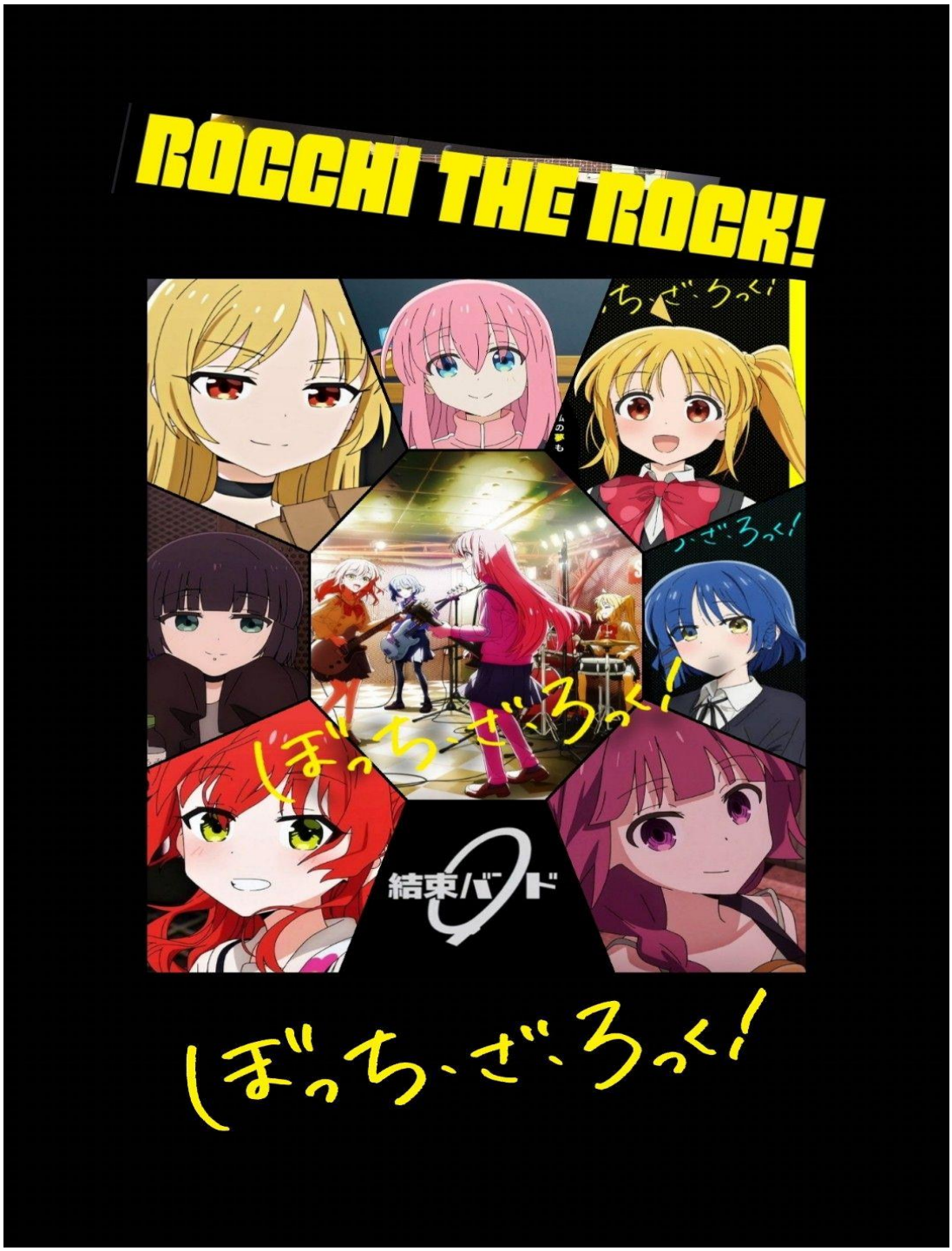
parte 2

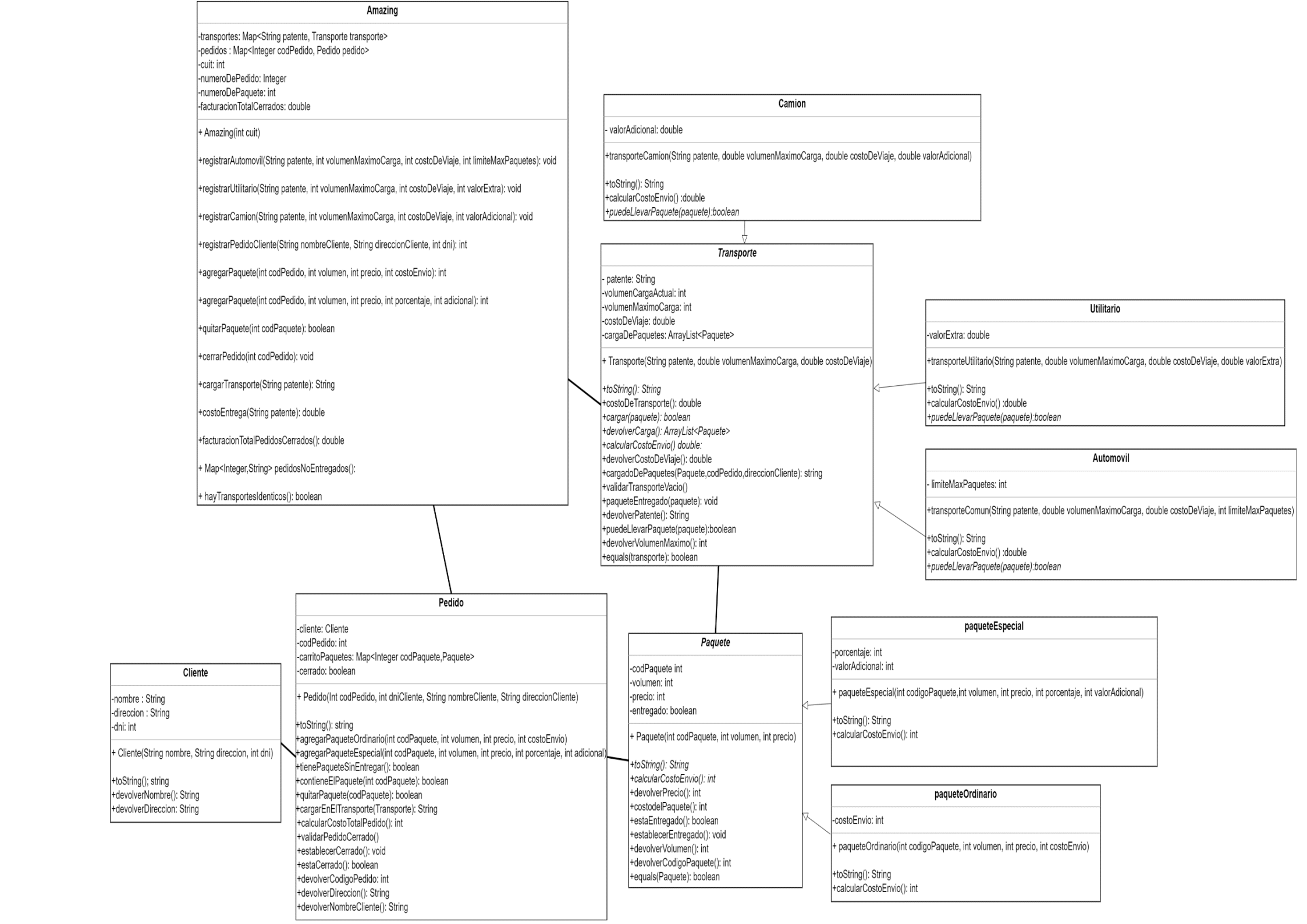
Alumnos: Cristian Leonel Jurajuria

Comisión: 03

Grupo: 10

Año: 2023




Amazing
-transportes: Map<String patente, Transporte transporte>
-pedidos : Map<Integer codPedido, Pedido pedido>
-cuit: int
-numeroDePedido: Integer
-numeroDePaquete: int
-facturacionTotalCerrados: double
+ Amazing(int cuit)
+registrarAutomovil(String patente, int volumenMaximoCarga, int costoDeViaje, int limiteMaxPaquetes): void
+registrarUtilitario(String patente, int volumenMaximoCarga, int costoDeViaje, int valorExtra): void
+registrarCamion(String patente, int volumenMaximoCarga, int costoDeViaje, int valorAdicional): void
+registrarPedidoCliente(String nombreCliente, String direccionCliente, 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): void
+cargarTransporte(String patente): String
+costoEntrega(String patente): double
+facturacionTotalPedidosCerrados(): double
+ Map<Integer,String> pedidosNoEntregados():
+ hayTransportesIdenticos(): boolean
Camion
- valorAdicional: double
+transporteCamion(String patente, double volumenMaximoCarga, double costoDeViaje, double valorAdicional)
+toString(): String
+calcularCostoEnvio() :double
+puedeLlevarPaquete(paquete):boolean
Transporte
- patente: String
-volumenCargaActual: int
-volumenMaximoCarga: int
-costoDeViaje: double
-cargaDePaquetes: ArrayList<Paquete>
+ Transporte(String patente, double volumenMaximoCarga, double costoDeViaje)
+toString(): String
+costoDeTransporte(): double
+cargar(paquete): boolean
+devolverCarga(): ArrayList<Paquete>
+calcularCostoEnvio() double:
+devolverCostoDeViaje(): double
+cargadoDePaquetes(Paquete,codPedido,direccionCliente): string
+validarTransporteVacio()
+paqueteEntregado(paquete): void
+devolverPatente(): String
+puedeLlevarPaquete(paquete):boolean
+devolverVolumenMaximo(): int
+equals(transporte): boolean
Utilitario
-valorExtra: double
+transporteUtilitario(String patente, double volumenMaximoCarga, double costoDeViaje, double valorExtra)
+toString(): String
+calcularCostoEnvio() :double
+puedeLlevarPaquete(paquete):boolean
Automovil
- limiteMaxPaquetes: int
+transporteComun(String patente, double volumenMaximoCarga, double costoDeViaje, int limiteMaxPaquetes)
+toString(): String
+calcularCostoEnvio() :double
+puedeLlevarPaquete(paquete):boolean
Pedido
-cliente: Cliente
-codPedido: int
-carritoPaquetes: Map<Integer codPaquete,Paquete>
-cerrado: boolean
+ Pedido(Int codPedido, int dniCliente, String nombreCliente, String direccionCliente)
+toString(): string
+agregarPaqueteOrdinario(int codPaquete, int volumen, int precio, int costoEnvio)
+agregarPaqueteEspecial(int codPaquete, int volumen, int precio, int porcentaje, int adicional)
+tienePaqueteSinEntregar(): boolean
+contieneElPaquete(int codPaquete): boolean
+quitarPaquete(codPaquete): boolean
+cargarEnElTransporte(Transporte): String
+calcularCostoTotalPedido(): int
+validarPedidoCerrado()
+establecerCerrado(): void
+estaCerrado(): boolean
+devolverCodigoPedido: int
+devolverDireccion(): String
+devolverNombreCliente(): String
Cliente
-nombre : String
-direccion : String
-dni: int
+ Cliente(String nombre, String direccion, int dni)
+toString(); string
+devolverNombre(): String
+devolverDireccion: String
Paquete
-codPaquete int
-volumen: int
-precio: int
-entregado: boolean
+ Paquete(int codPaquete, int volumen, int precio)
+toString(): String
+calcularCostoEnvio(): int
+devolverPrecio(): int
+costodelPaquete(): int
+estaEntregado(): boolean
+establecerEntregado(): void
+devolverVolumen(): int
+devolverCodigoPaquete(): int
+equals(Paquete): boolean
paqueteEspecial
-porcentaje: int
-valorAdicional: int
+ paqueteEspecial(int codigoPaquete,int volumen, int precio, int porcentaje, int valorAdicional)
+toString(): String
+calcularCostoEnvio(): int
paqueteOrdinario
-costoEnvio: int
+ paqueteOrdinario(int codigoPaquete, int volumen, int precio, int costoEnvio)
+toString(): String
+calcularCostoEnvio(): int

IREPS

Clase Amazing

Datos:
-cuit
-transportes
-pedidos
-numeroDePedido
-numeroDePaquete
-facturacionTotalCerrados

Irep:
-cuit debe no ser vacio.
-transportes es un map del objeto Transporte, tampoco pueden tener valores null.
-pedidos es un map del objeto Pedido, tampoco pueden tener valores null.
-numeroDePedido debe ser único.
-numeroDePaquete debe ser único.
-facturacionTotalCerrados debe ser >= 0, !=null, es igual a la suma del costo de todos los pedidos cerrados

Clase Pedido

Datos:
-Cliente
-codPedido
-carritoDePaquetes
-cerrado

Irep:
-Cliente no debe ser nulo, cada pedido debe estar asociado a un cliente valido.
-codPedido debe ser >= 0 y es único.
-carritoDePaquetes debe contener objetos de tipo Paquete y no debe ser nula.
-cerrado debe ser booleano e indica si el pedido está o no cerrado.

Clase Cliente

Datos:
-Nombre
-Direccion
-DNI

Irep:
-Nombre no debe estar vacio ni nulo.
-Dirección no debe estar vacio ni nulo.
-DNI debe tener 8 numeros.

Clase Transporte

Datos:
-Patente
-volumenDeCargaActual
-costoDeViaje
-volumenMaximoCarga
-cargaDePaquetes

Irep:
-Patente no debe estar vacía ni nulo.
-costoDeViaje debe ser > =0
-volumenMaximoCarga debe ser > =0 y además debe ser mayor o igual a la suma del volumen de los paquetes cargados. Menos o igual que el volumenDeCargaActual.
-volumenDeCargaActual debe mantenerse entre 0 y el volumenMaximoCarga. También es igual al volumen de los paquetes cargados hasta el momento.
-cargaDePaquetes no debe ser nula y contiene objetos de tipo Paquete

Clase Camión

Datos:
-valorAdicional

Irep:
-valorAdicional >=0 y su valor se multiplica por la cantidad de paquetes del transporte.
-Paquetes solo lleva paquetes especiales y con volumen >2000

Clase Automovil

Datos:
-limiteMaxPaquetes

Irep:
-limiteMaxPaquetes debe ser > =0, no puede superar su límite y es mayor a la cantidad de paquetes cargados.
-Paquetes deben ser ordinarios y con volumen <2000

Clase Utilitario

Datos:

-valorExtra

Irep:
-valorExtra debe ser > =0

Clase Paquete

Datos:
-entregado
-codPaquete
-volumen
-precio

Irep:
-codPaquete debe ser >= 0 y único.
-volumen debe ser > =0
-precio > =0
-entregado valor booleano indica si el paquete ha sido entregado o no.

Clase paqueteEspecial

Datos:
-porcentaje
-valorAdicional

Irep:
-porcentaje debe ser > 0 y aumenta el costo base con dicho porcentaje.
-valorAdicional debe ser >= 0 y se suma en caso de que el paquete tenga volumen >=3000.

Clase paqueteOrdinario

Datos:
-costoEnvio

Irep:
-costoEnvio debe ser >= 0

Requisitos del enunciado cumplidos:

Utilizamos los siguientes conceptos: Herencia, polimorfismo, sobreescritura, sobrecarga, clases/métodos abstractos.

Un Ejemplo que cumple con estos casos menos la sobrecarga es el método de la clase abstracta Paquete calcularCostoEnvio.

Tenemos herencia en paqueteEspecial y paqueteOrdinario ya que heredan la clase abstracta Paquete. Tambien tenemos herencia en Utilitario, Automovil y camion ya que heredan la clase abstracta Transporte.

Luego tenemos polimorfismo, ya que tenemos el método calcularCostoEnvio que en las clases paqueteEspecial y paqueteOrdinario, actúan de forma diferente en cada clase. De igual manera también tenemos método calcularCostoEnvio pero aplicado en los diferentes tipos de transporte, para asi también proporcionar el calculo de costo de un transporte dependiendo los requisitos y condiciones de cada transporte. Tenemos diversos, se han utilizado en este trabajo.

Por otro lado, tenemos sobreescritura dentro de paqueteEspecial y paqueteOrdinario, ya que sobrescribimos el método calcularCostoEnvio heredado de la clase Paquete para proporcionar su propia lógica. Tenemos sobreescritura también en lo que son los toString de cada tipo de paquete ya que queremos que cada paquete imprima no solo los datos generales sino los particulares de ellos mismos. Esto también se repite para los toString de cada tipo de transporte. Tambien contamos con el método puedellevarPaquete(Paquete paquete) el cual esta en transporte pero se sobreescribe tanto en camion como en automóvil y en utilitario, básicamente cada transporte que llame ese método va a tener su propia lógica para saber si puede llevar un determinado tipo de paquete o no. Y tenemos varios mas.

Para cumplir la sobrecarga, podemos agarrar el método agregarPaquete de la clase EmpresaAmazing, ya que contamos con dos versiones del método agregarPaquete que aceptan diferentes parámetros, uno para paquetes ordinario que toma como parámetro (codPedido, volumen, precio, costoEnvio) y otro para los paquetes especiales que toma como parámetro (codPedido, volumen, precio, porcentaje, adicional).

Tecnologías Java:

StringBuilder:
Utilizamos el StringBuilder en el método cargarTransporte de la clase EmpresaAmazing, ya que cada vez que cargamos un transporte, lo cargamos con paquete y utilizando la tecnología de java podemos armar una lista de paquetes cargados que se podrá ir actualizando en formato String.

Iteradores y Foreach:

Utilizamos un Iterador y Foreach en el método quitarPaquete de la clase EmpresaAmazing, ya que primero recorremos los pedidos (Foreach) y luego aplicamos un iterador para obtener la colección de paquetes y esto nos garantiza buscar el paquete que se desea eliminar más fácilmente. Este es un ejemplo puntual pero los Foreach los utilizamos casi a lo largo de todo el tp.

Complejidad y Algebra de Ordenes:

Ejercicio 4): quitarPaquete ()

```
        @Override
        public boolean quitarPaquete(int codPaquete) {
O(n*m)        if (buscarPaquete(codPaquete) == null) {
            throw new RuntimeException("No se encontró el paquete");
        }
O(n*m)       if (buscarPedidoConCodPaquete(codPaquete).estaCerrado()) {
            throw new RuntimeException("El pedido está cerrado");
        }

O(n)        for (Pedido pedido : pedidos.values()) {
O(1)            Iterator<Paquete> iterator = pedido.devolverCarritoPaquetes().iterator();
O(n)            while (iterator.hasNext()) {
O(1)                Paquete paquete = iterator.next();
O(1)                if (paquete.devolverCodigoPaquete() == codPaquete) {
O(1)                    pedido.eliminarDelCarrito(paquete);
O(1)                    return true;
                }
            }
        }

O(1)        return false;
    }
```

```
Total=  O(n*m) + O(n*m) + O(n) + O(1) + O(n) + O(1) + O(1) + O(1) + O(1) + O(1)
         =      O(n*m) + O(n*m) + O(n) + O(n)
         =      O(n*m) + O(n) + O(n)    //Regla 2
         =      O(n*m) + O(n+n)                 //Regla 2
         =      O(n*m + 2n)                             //Regla 2
         =      O(n*m) + O(2n)                  //Regla 2
         =      O(n*m) + O(2) . O(n)    //Regla 3
         =      O(n*m) + O(1) . O(n)    //Regla 4
         =      O(n*m) + O(n)                   //Regla 3
         =      O(n*m + n)                              //Regla 2
         =      O(n(m+1)                                //FactorComun N
         =      O(n*m)
```

```
            private Paquete buscarPaquete(int codigoPaquete) {
        {       for (Pedido pedido : pedidos.values()) {
O(n*m){
        {       for (Paquete paquete : pedido.devolverCarritoPaquetes()) {
O(1)                            if (paquete.devolverCodigoPaquete() == codigoPaquete) {
O(1)                                    return paquete;
                                        }
                                }
                        }
O(1)            return null;
                }

Total=  O(n*m)+O(1)+O(1)+O(1)
     =  O(n*m)



        private Pedido buscarPedidoConCodPaquete(int codigoPaquete) {
        {       for (Pedido pedido : pedidos.values()) {
O(n*m){
        {       for (Paquete paquete : pedido.devolverCarritoPaquetes()) {
O(1)                            if (paquete.devolverCodigoPaquete() == codigoPaquete) {
O(1)                                    return pedido;
                                }
                        }
                }
O(1)            return null;

        }

Total=  O(n*m)+O(1)+O(1)+O(1)
     =  O(n*m)
```