PROGRAMACION 2: EMPRESA AMAZING

Docentes:

Martha Semken y Karin Fleischer

Alumnos:

Ignacio Aranda Bao y Nahuel Herrera

Programación II - Trabajo Práctico Integrador 2do. Cuatrimestre 2023.

U.N.G.S(Universidad Nacional De General Sarmiento)

Programación II.

Nombre de las docentes a cargo: Martha Semken, Karin Fleischer

Nombres de los integrantes del grupo: Aranda Bao Ignacio, Herrera Nahuel.

Informe:

Durante el desarrollo de este proyecto, nos enfrentamos a diversos desafíos que surgieron a medida que traducíamos el concepto de los Tipos Abstractos de Datos (TADs) a código. Inicialmente, teníamos una comprensión clara de los TADs, pero al consultar con las docentes, identificamos un problema significativo con el TAD Cliente. Este TAD estaba asumiendo responsabilidades que no le correspondían, como se evidencia en la primera parte del trabajo (se adjunta el archivo correspondiente).

Para abordar este problema, realizamos modificaciones en el código redistribuyendo las responsabilidades, donde el TAD Pedido asumió varias funciones que previamente pertenecían al TAD Cliente. Ahora, el TAD Cliente se limita a almacenar datos sin manipular otras acciones mas que las consultas de sus propios datos. Además, optamos por el uso de herencia en la clase Transporte, que es abstracta y hereda datos y acciones a las clases TransporteComun, TransporteCamion y TransporteUtilitario.

Herencia de Transporte y abstraccion del mismo:

```
public abstract class Transporte {
    private String Patente;
    private int VolumenMaximoPorCarga;
    private double CostoPorViaje;
    private ArrayList<Paquete> paquetes;
    private int VolumenActual;


    public Transporte() {}

    public Transporte(String patente, int volumenMaximoPorCarga, int costoPorViaje) {
        Patente = patente;
        VolumenMaximoPorCarga = volumenMaximoPorCarga;
        CostoPorViaje = costoPorViaje;
        this.paquetes = new ArrayList<Paquete>();
        VolumenActual=0;
    }
```

```java
public class TransporteUtilitario extends Transporte{
    private ArrayList <Paquete> paquetesOrdYEsp;
    private double ValorExtra;

    public TransporteUtilitario(String patente, int volumenMaximoPorCarga, int costoPorViaje, int valorExtra) {
        super(patente, volumenMaximoPorCarga, costoPorViaje);
        this.ValorExtra= valorExtra;
        this.paquetesOrdYEsp= new ArrayList<Paquete>();
    }
```

```java
public class TransporteComun extends Transporte {
    private ArrayList <Paquete> paquetesOrdinarios;
    private int MaxPaquetes;

    public TransporteComun(String patente, int volumenMaximoPorCarga, int costoPorViaje, int maxPaquetes) {
        super(patente, volumenMaximoPorCarga, costoPorViaje);
        this.paquetesOrdinarios= new ArrayList<Paquete>();
        this.MaxPaquetes= maxPaquetes;
    }
```

```java
public class TransporteCamion extends Transporte {
    private ArrayList<Paquete> PaquetesEspeciales;
    private double ValorExtra;


    public TransporteCamion(String patente, int volumenMaximoPorCarga, int costoPorViaje, int valorExtra) {
        super(patente, volumenMaximoPorCarga, costoPorViaje);
        this.ValorExtra = valorExtra;
        this.PaquetesEspeciales = new ArrayList<Paquete>();
        this.getVolumenActual();
    }
```

En cada subclase, implementamos la sobreescritura de métodos, especialmente en cargarPaquete() y calcularCostoDeViaje(), los cuales son heredados de la superclase Transporte y se utilizan de manera diferente en cada subclase. También implementamos getters y setters, así como el método toString(). En la clase Empresa, empleamos el polimorfismo con los métodos registrarAutomovil(), registrarUtilitario() y registrarCamion(), ya que Transporte asume diferentes formas según los parámetros proporcionados.

Sobreescritura de cargarPaquete:

```java
abstract public void cargarPaquete(Paquete paquete) ;
```

```java
@Override
public void cargarPaquete(Paquete paquete) {
    if (paquete==null) {
        throw new RuntimeException("Paquete es null");
        }
    if(this.getVolumenActual()+paquete.getVolumen()<this.getVolumenMaximoPorCarga()) {
        if (!paqueteRepetido(paquete) && esEspecial(paquete) && (paquete.getVolumen() >2000) ){
            this.PaquetesEspeciales.add(paquete);
            this.aumentarVolumen(paquete.getVolumen());
            paquete.setEntregado(true);
            }
        }
    }

@Override
public void cargarPaquete(Paquete paquete) {
    if (paquete==null) {
        throw new RuntimeException("Paquete es null");
        }

    if(this.getVolumenActual()+paquete.getVolumen()<this.getVolumenMaximoPorCarga()||paquetesOrdinarios.size()>MaxPaquetes) {

        if (!paqueteRepetido(paquete) && esOrdinario(paquete) && (paquete.getVolumen() <= 2000) ){
            this.paquetesOrdinarios.add(paquete);
            this.aumentarVolumen(paquete.getVolumen());
            paquete.setEntregado(true);
        }
    }
}
```

Posteriormente, aplicamos la herencia en la superclase Paquete, la cual es abstracta y hereda a las clases PaqueteOrdinario y PaqueteEspecial. Utilizamos la sobreescritura en los métodos mismoPaquete() y precioFinalDePaquete(). El método equals() es abstracto y presenta polimorfismo al funcionar de manera diferente en cada subclase.

En el TAD Empresa, incorporamos polimorfismo y sobrecarga en el método agregarPaquete(), que comparte el mismo nombre pero diferentes parámetros. Dependiendo de los valores proporcionados, el paquete puede ser Ordinario o Especial.

Polimorfismo y sobrecarga de agregarPaquete():

```java
public int agregarPaquete(int codPedido, int volumen, int precio, int porcentaje, int adicional) {

    Pedido pedido = obtenerPedido(codPedido);
    return pedido.agregarPaqueteEsp(codPedido, volumen, precio, porcentaje, adicional);
}

public int agregarPaquete(int codPedido, int volumen, int precio, int costoEnvio) {
    Pedido pedido = obtenerPedido(codPedido);
    return pedido.agregarPaqueteOrd(codPedido, volumen, precio, costoEnvio);
    }
```

Destacamos la elección de utilizar dos diccionarios (Transporte y Pedido) en la clase Empresa, considerándolos como estructuras de datos eficientes para obtener rápidamente la información requerida. Además, implementamos acumuladores booleanos en el método equals() de Paquete que a

su vez esta sobreescrito y utilizamos StringBuilder en el método cargarTransporte().

Acumuladores booleanos de equals():

```
@Override
public boolean equals(Paquete paquete) {
    boolean acum=true;
    if(paquete instanceof PaqueteEspecial) {
        PaqueteEspecial paqueteEsp =(PaqueteEspecial) paquete;
            acum=acum && this.getVolumen()==paqueteEsp.getVolumen() &&
                this.getPrecio()==paqueteEsp.getPrecio()&&
                this.getPorcentaje()==paqueteEsp.getPorcentaje()&&
                this.getAdicional()==paqueteEsp.getAdicional();
    }
    else {
        return false;
    }
    return acum;
}
```

A lo largo del desarrollo, nos encontramos con complejidades, como problemas en los testeos que revelaron errores en ciertos métodos. También hubo desafíos en la correcta interpretación de las consignas, siendo la última un problema de encapsulamiento y mal manejo de los datos en la clase Empresa. Esta dificultad se resolvió evitando sobrecargar la clase con responsabilidades y respetando las firmas requeridas por la interfaz.

En conclusión, a pesar de las complejidades encontradas, el proyecto nos proporcionó valiosas herramientas en el manejo de objetos. La organización de los TADs y la elaboración del Diagrama facilitaron la implementación del código de manera ordenada.

Complejidad del metodo solicitado:

```java
    // la complejidad es O(N al cuadrado)
    public boolean eliminarPaqueteDelCarrito(int codPaquete) {
        Paquete paquete = obtenerPaquetePorCodigo(codPaquete);       //O(N) + O(1)
        if (paquete != null) {                                       //O(1)
            return this.Carrito.remove(paquete);                     //O(N)
        }
        return false;
    }
    //metodo auxilliar de eliminarPaquete
    private Paquete obtenerPaquetePorCodigo(int codPaquete) {
        for (Paquete paquete : this.Carrito) {           //O(N)
            if (paquete.getIdentificador() == codPaquete) {      // O(1)
                return paquete; // Se encontró el paquete        O(1)
            }
        }
        return null;
    }

```

```java
    //el metodo quitarPaquete es de O(n cubo .k)
public boolean quitarPaquete(int codPaquete) {
        Pedido pedido = obtenerpedidoConUnPaquete(codPaquete);      // O(1)+(obtenerpedidoConUnPaquete tiene O(n.k))
        return pedido.eliminarPaqueteDelCarrito(codPaquete);        // O(N cuadrado)

    }
// Método auxiliar para obtener el pedido por código de paquete cuyo orden es O(n.k), con 'n' siendo el largo de pedidosRegistrados y 'k el largo del carrito del pedido
public Pedido obtenerpedidoConUnPaquete(int codPaquete) {
        ArrayList<Pedido> pedidosTotales = new ArrayList<>(this.pedidosRegistrados.values());
        for(Pedido pedidoActual: pedidosTotales) {          //O(n)
            for(Paquete PaqueteActual: pedidoActual.getCarrito()) { //O(k)
                if (PaqueteActual.getIdentificador() == codPaquete) {   //O(1)
                    return pedidoActual;
                }
            }
        }
        return null;
    }

```

TADS IREP Y FIRMA DE MÉTODOS:

TAD Empresa:

Datos:

-String CUIL //debe ser no nulo

-Diccionario Transporte flota

-Diccionario Pedido PedidosRegistrados

Acciones:

+void registrarAutomovil(String Patente, int volMax, int valorViaje, int maxPaquete){ }//agrega un transporteComun a flota de transportes

+void registrarUtilitario(String Patente, int volMax, int valorViaje, int adicXPaq){ }//agrega un transporteUtilitario a flota de transportes

+void registrarCamion(String Patente, int volMax, int valorViaje, int valorExtra){ }//agrega un transporteCamion a flota de transportes

+void registrarPedido(String cliente, String direccion, Int DNI){}//agrega un pedido diccionario pedidos

+int agregarPaquete(int codPedido, int volumen, int precio, int porcentaje, int adicional){}//agrega un PaqueteEspecial al carrito de pedidos

+int agregarPaquete(int codPedido, int volumen, int precio, int costoEnvio){}//agrega un PaqueteOrdinario al carrito de pedidos

+boolean quitarPaquete{}(int codPaquete)

+double cerrarPedido(int codPedido){}//cierra el pedido y devuelve el total a pagar del mismo

+double facturacionTotalPedidosCerrados(){}//devuelve la facturacion total de todos los pedidos cerrados

+boolean HayTransporteIdenticos(){}//recorre la flota de transportes buscando que haya transportes identicos con la condicion preestablecida en en el enunciado

+double costoEntrega(String patente){}//devuelve el costo de entrega del transporte que tenga la patente pasada por parametro

+String cargarTransporte(String patente){}//si pedido esta finalizado se carga el transporte con la patente pasada por parametro con paquetes que no esten entregados, luego se devuelve un StringBuilder convertido a String

TAD Cliente :

Datos:

-String Nombre //debe ser no nulo

-String Dirección //debe ser no nulo

-Int DNI (unico) //debe ser no nulo y no puede estar repetido

TAD Pedido:

Datos:

-int NúmeroDePedido //unico y no nulo

-Static int contadorNumeroDePedido=1

-Cliente cliente //debe ser no nulo

-boolean Finalizado //comienza en false

-boolean Entregado //comienza en false

-ArrayList <Paquete> Carrito

Acciones:

+boolean yaEstaEnCarrito(Paquete producto){}

+void agregarProductoAlCarrito(Paquete paquete){}

+boolean eliminarPaqueteDelCarrito(int codPaquete){}

+boolean tienePaquete(int codPaquete){}

TAD Paquete :

Datos:

-int Identificador // único y no nulo

-Static int contadorNumeroDePaquete =1

-Int Volumen // debe ser > = 0 y no nulo

-int Precio // debe ser > = 0 y no nulo

-Boolean Entregado // empieza en false

Acciones:

+static int obtenerNumeroPaqueteUnico()

+double costoDePaquete()

+abstract double precioFinalDePaquete()

TAD Paquete Ordinario : Hereda de paquete

Datos:

-int costoDeEnvio // debe ser > = 0 y no nulo

Acciones:

+double precioFinalDePaquete{}//

+boolean mismoPaquete(Paquete paquete){}//

TAD Paquete Especial: Hereda de paquete

Datos:

-int Porcentaje //debe ser >= 0 y no nulo

-int Adicional //debe ser >= 0 y no nulo

Acciones:

+int calcularPorcentaje(){}//

+double precioFinalDePaquete{}//

+boolean mismoPaquete(Paquete paquete){}//

TAD Transporte:

Datos:

-String Patente // debe ser unico y no nulo

-Int VolumenMáximoPorCarga // debe ser >= 0 y no nulo

-Double CostoPorViaje // debe ser >= 0 y no nulo

-ArrayList Paquete Paquetes

-int volumenActual // debe ser >= 0 y no nulo

Acciones:

+boolean paqueteRepetido(Paquete paquete){}

+abstract void cargarPaquete(Paquete paquete){}

+abstract double calcularCostoDeViaje(){}

+void aumentarVolumen(int volumenDePaquete){}

+ArrayList Paquete PaqueteCargados(){}

TAD Transporte Comun: hereda de transporte

Datos:

-ArrayList Paquete paquetesOrdinarios

-int maxPaquetes // debe ser >= 0 y no nulo

Acciones:

+void cargarPaquete (Paquete paquete){}

+double calcularCostoDeViaje(){}

TAD Transporte Utilitario: hereda de transporte

Datos:

-ArrayList Paquete paquetesOrdYEsp

Acciones:

+void cargarPaquete (Paquete paquete){}

+double calcularCostoDeViaje(){}

TAD Transporte Camion: hereda de transporte

Datos:

-ArrayList Paquete PaquetesEspeciales

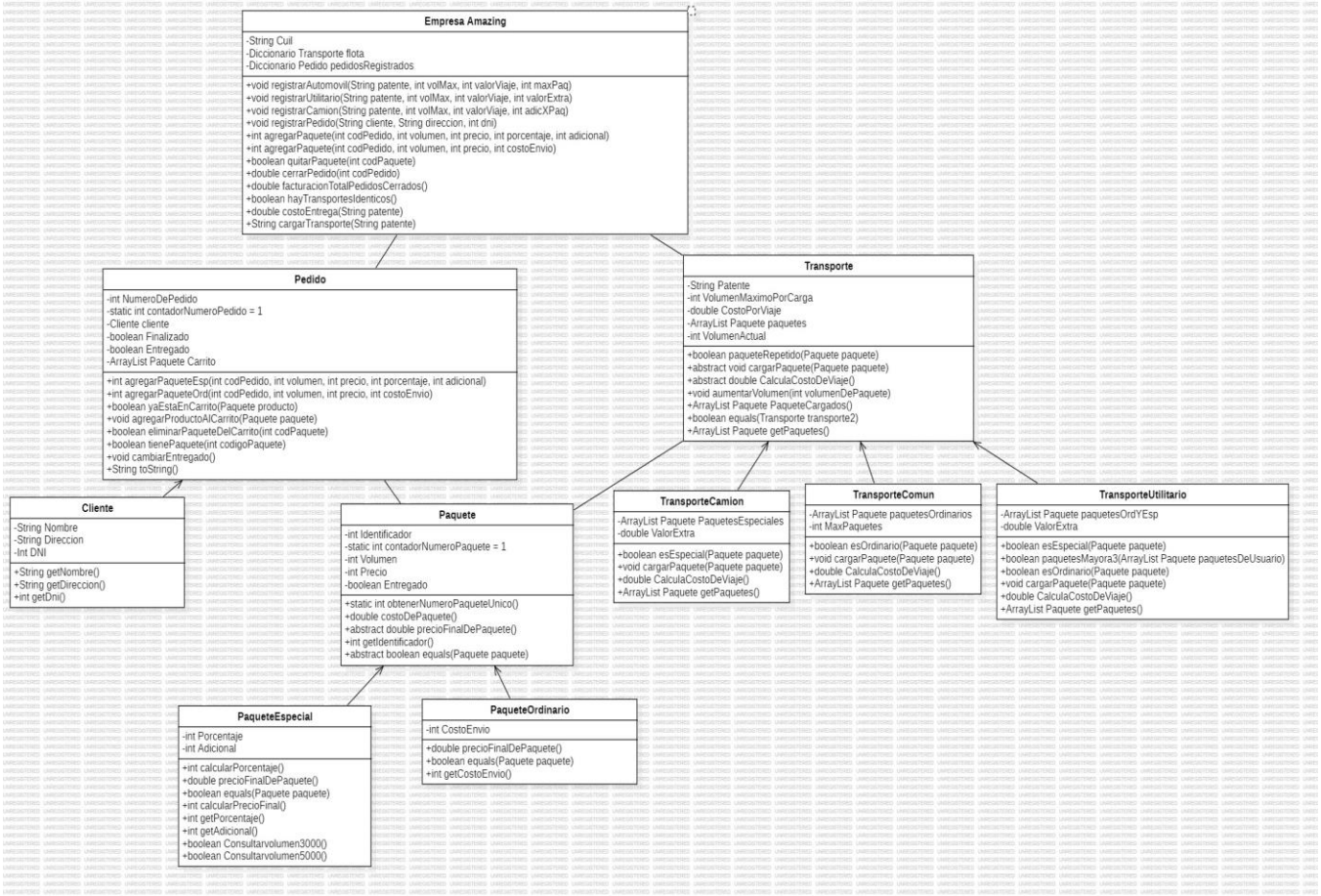
-Double ValorExtra // debe ser >= 0 y no nulo

Acciones:

+void cargarPaquete (Paquete paquete){}

+double calcularCostoDeViaje(){}

DIagrama:


Empresa Amazing
-String Cuil
-Diccionario Transporte flota
-Diccionario Pedido pedidosRegistrados
+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 porcentaje, int adicional)
+int agregarPaquete(int codPedido, int volumen, int precio, int costoEnvio)
+boolean quitarPaquete(int codPaquete)
+double cerrarPedido(int codPedido)
+double facturacionTotalPedidosCerrados()
+boolean hayTransportesIdenticos()
+double costoEntrega(String patente)
+String cargarTransporte(String patente)
Pedido
-int NumeroDePedido
-static int contadorNumeroPedido = 1
-Cliente cliente
-boolean Finalizado
-boolean Entregado
-ArrayList Paquete Carrito
+int agregarPaqueteEsp(int codPedido, int volumen, int precio, int porcentaje, int adicional)
+int agregarPaqueteOrd(int codPedido, int volumen, int precio, int costoEnvio)
+boolean yaEstaEnCarrito(Paquete producto)
+void agregarProductoAlCarrito(Paquete paquete)
+boolean eliminarPaqueteDelCarrito(int codPaquete)
+boolean tienePaquete(int codigoPaquete)
+void cambiarEntregado()
+String toString()
Transporte
-String Patente
-int VolumenMaximoPorCarga
-double CostoPorViaje
-ArrayList Paquete paquetes
-int VolumenActual
+boolean paqueteRepetido(Paquete paquete)
+abstract void cargarPaquete(Paquete paquete)
+abstract double CalculaCostoDeViaje()
+void aumentarVolumen(int volumenDePaquete)
+ArrayList Paquete PaqueteCargados()
+boolean equals(Transporte transporte2)
+ArrayList Paquete getPaquetes()
Cliente
-String Nombre
-String Direccion
-Int DNI
+String getNombre()
+String getDireccion()
+int getDni()
Paquete
-int Identificador
-static int contadorNumeroPaquete = 1
-int Volumen
-int Precio
-boolean Entregado
+static int obtenerNumeroPaqueteUnico()
+double costoDePaquete()
+abstract double precioFinalDePaquete()
+int getIdentificador()
+abstract boolean equals(Paquete paquete)
TransporteCamion
-ArrayList Paquete PaquetesEspeciales
-double ValorExtra
+boolean esEspecial(Paquete paquete)
+void cargarPaquete(Paquete paquete)
+double CalculaCostoDeViaje()
+ArrayList Paquete getPaquetes()
TransporteComun
-ArrayList Paquete paquetesOrdinarios
-int MaxPaquetes
+boolean esOrdinario(Paquete paquete)
+void cargarPaquete(Paquete paquete)
+double CalculaCostoDeViaje()
+ArrayList Paquete getPaquetes()
TransporteUtilitario
-ArrayList Paquete paquetesOrdYEsp
-double ValorExtra
+boolean esEspecial(Paquete paquete)
+boolean paquetesMayora3(ArrayList Paquete paquetesDeUsuario)
+boolean esOrdinario(Paquete paquete)
+void cargarPaquete(Paquete paquete)
+double CalculaCostoDeViaje()
+ArrayList Paquete getPaquetes()
PaqueteEspecial
-int Porcentaje
-int Adicional
+int calcularPorcentaje()
+double precioFinalDePaquete()
+boolean equals(Paquete paquete)
+int calcularPrecioFinal()
+int getPorcentaje()
+int getAdicional()
+boolean Consultarvolumen3000()
+boolean Consultarvolumen5000()
PaqueteOrdinario
-int CostoEnvio
+double precioFinalDePaquete()
+boolean equals(Paquete paquete)
+int getCostoEnvio()