Universidad Nacional de General Sarmiento

Programación II - Trabajo Práctico Integrador 2do. Cuatrimestre 2023

SEGUNDA PARTE

Alumnos: Lucas Quintana y Ciro Lopez.

Comisión: 03.

Profesores: José Nores - Adrián Cáceres

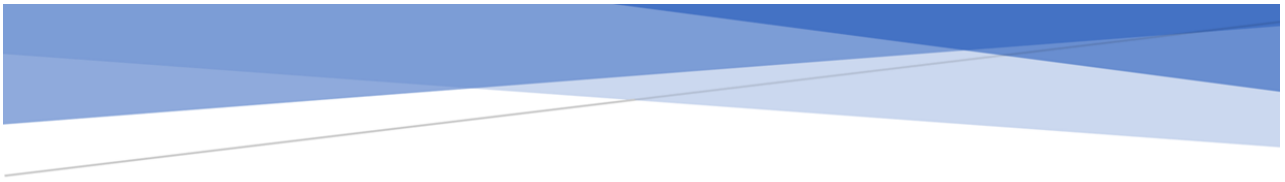
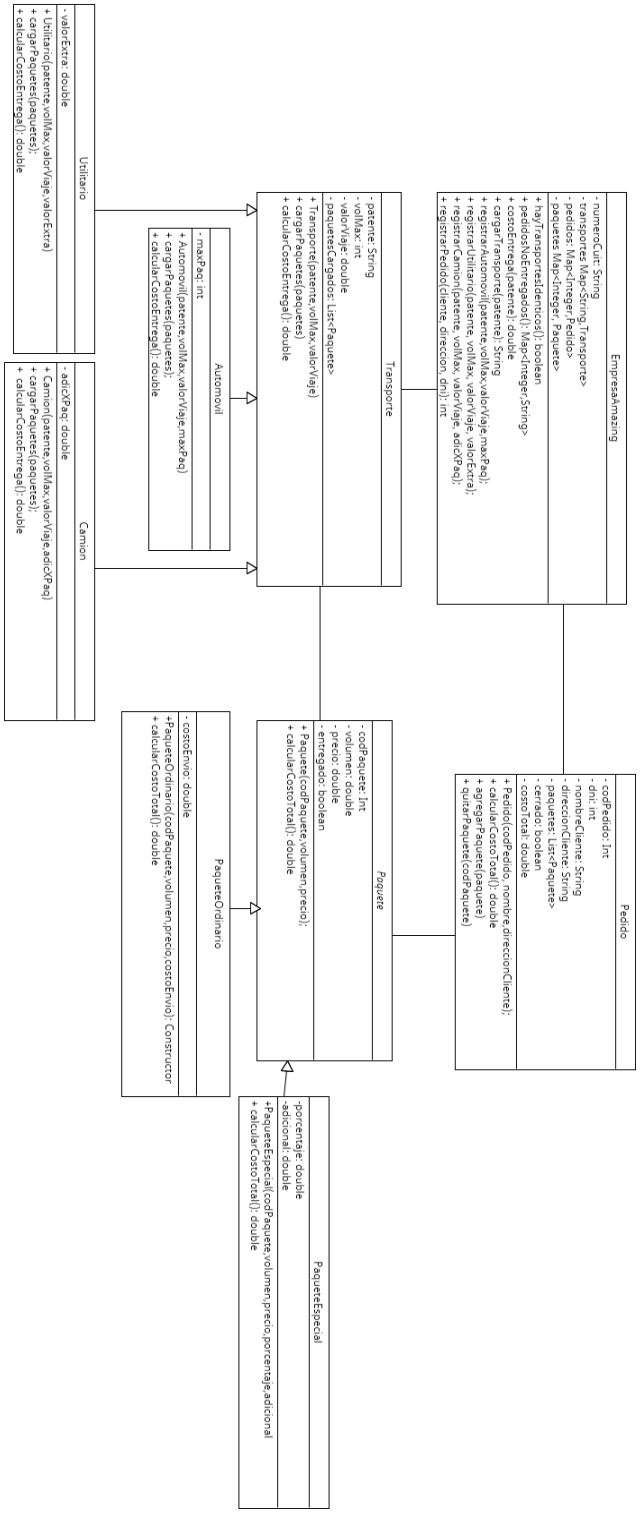


Diagrama De Clase



IREP para cada tipo de dato:

Empresa Amazing:

- El atributo numeroCuit no debe ser nulo y debe ser una cadena no vacía.
- El mapa transportes no debe ser nulo y debe contener entradas válidas de patentes y objetos de tipo Transporte.
- La lista pedidos no debe ser nula y debe contener instancias válidas de la clase Pedido.
- El valor de codPaquete debe ser un número entero no negativo.
- En el atributo transporte, cuando se registra un nuevo transporte, la patente no tiene que estar duplicada en el mapa de transportes.
- En el atributo pedido, cuando se registra un nuevo pedido, el código del pedido debe ser único y mayor o igual a 10.

Pedido:

- El atributo codPedido debe ser un número entero no negativo.
- El atributo dni debe ser un número entero no negativo.
- El atributo nombreCliente no debe ser nulo y debe ser una cadena no vacía.
- El atributo direccionCliente no debe ser nulo y debe ser una cadena no vacía.
- La lista de paquetes no debe ser nula y debe contener instancias válidas de la clase Paquete.
- El atributo cerrado debe ser verdadero (true) si el pedido está cerrado y falso (false) si no lo está.
- El atributo costoTotal debe ser un número no negativo.

  Paquete:
- El atributo id debe ser un número entero no negativo.
- El atributo volumen debe ser un número entero no negativo.
- El atributo precio debe ser un número no negativo.
- El atributo codPedido debe ser un número entero no negativo.
- El atributo entregado debe ser verdadero (true) si el paquete está entregado y falso (false) si no lo está.
- El atributo pedido debe ser una referencia válida a una instancia de la clase Pedido.

PaqueteOrdinario:

- El atributo costoEnvio debe ser un número no negativo.

PaqueteEspecial:

- Los atributos valorAdicional y porcentajeAdicional deben ser números no negativos.

Transporte:

- El atributo patente no debe ser nulo ni una cadena vacía.
- El atributo volMax debe ser un número positivo.
- La lista paquetesCargados debe estar inicializada correctamente.
- la lista paquetesCargados no puede ser mayor al volumen máximo
- la lista paquetesCargados tiene que estar en la lista de paquetes

Automovil:

- El atributo maxPaq debe ser un número positivo.
- Tiene que tener como máximo posible la carga de paquetes (paquetesCargados) el volMax correspondiente

Camion:

- El atributo adicXPaq debe ser un número positivo.

Utilitario:

- El atributo valorExtra debe ser un número no negativo.

Conceptos Utilizados:

- Sobrecargar: Es la capacidad de una clase de tener múltiples métodos con el mismo nombre en la misma clase, pero con diferentes parámetros.
  Utilizamos sobrecarga en el método agregarPaquete de la clase EmpresaAmazing.

- Sobreescritura:

  Utilizamos sobreescritura en los métodos de calcularCostoEntrega en las subclases de Transporte (Camion y Automovil), sobreescritura de toString en la clase Transporte y sus subclases (Camion y Automovil), sobreescritura de cargarTransporte en la clase Pedido, sobreescritura de calcularCostoTotalDelPaquete en las subclases de Paquete.

- Interfaz:.

  Utilizamos la interfaz IEmpresa para definir el conjunto de métodos que tendría que tener EmpresaAmazing.

- Métodos Abstractos:
  Utilizamos el método abstracto calcularCostoEntrega en Transporte que devuelve un double y también lo utilizamos el metodo abstracto calcularCostoTotalDelPaquete en Paquete que es un void.

- Clases Abstractas:
  Paquete es una clase abstracta al igual que Transporte.

- Herencia:
  Utilizamos herencia de la clase Transporte en las subclases camión, automóvil y utilitario, y lo utilizamos en paqueteOrdinario y paqueteEspecial que son subclase de la clase padre Paquete.

- Polimorfismo:
  Aproveché el polimorfismo en el método costoEntrega cuando se llama al método calcularCostoEntrega(), dependiendo el tipo de transporte cambia el resultado.

```java
@Override
public double costoEntrega(String patente) {
   validarExistenciaPatente(patente);
   Transporte transporte = obtenerTransporte(patente);
   return transporte.calcularCostoEntrega();
}
```

Explicación de la complejidad y Álgebra de Órdenes para el punto 4.

La complejidad del método es de O(n*m)

Complejidad:

quien es n? n= pedidos.length

quien es m? m= paquetes.length

```
// en el metodo quitarPaquete:
Public boolean quitarPaquete(int codPaquete){
for(Pedido pedido: pedidos){
        for(Paquete paquete: pedido.getPaquetes()){
                if(paquete.getId()==codPaquete){
                        Return pedido.quitarPaquete(codPaquete);
                }
        }
}
Throw new RuntimeException("El codigo de  paquete no está registrado.");
}
```

```
//en el metodo quitarPaquete de la clase pedido
Public boolean quitarPaquete(int codPaquete){
for(Paquete paq: paquetes){
        if(paq.getId()==codPaquete){
                paquetes.remove(paq);
                Return true;
        }
}
Return false;
}
```

Algebra de Órdenes:

O(n+m×n+1×n+m+1xm+m+1+1)

O(n+2m×n+2m+2)ORDENAMOS

O(max{n,2m×n,2m,2}) REGLA 2

O(max{n,m×n,m,1}) REGLA 4

Complejidad: O(m X n)

quien es n? n= pedidos.length

quien es m? m= paquetes.length

Rta= O(m*n)