PROGRAMACION II:

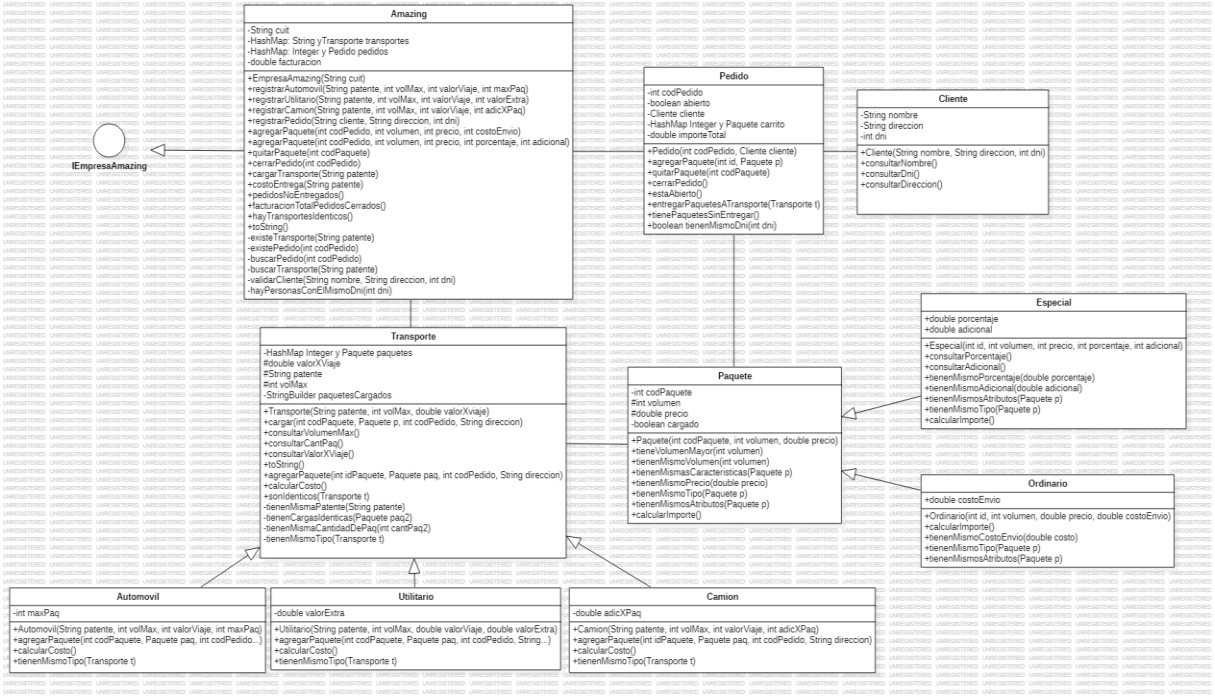
Segundo cuatrimestre 2023

Trabajo Práctico

Carolina Villalba

Comisión 1

//El HashMap de carrito en Pedido fue cambiado por LinkedList<Paquete>



III) El **polimorfismo** surgió por la necesidad de una clase o de un método de desarrollar distintos tipos de sí mismo. Por ejemplo, hay diferentes tipos de Tranporte, los cuales son Automovil, Utilitario y Camion. Lo mismo sucede con los Paquete que pueden ser Especial u Ordinario. En el caso de los métodos, se pueden mencionar aquellos cuya firma es la misma pero su implementación no lo es, como lo son las formas de calcular los precios.

En cuanto a las clases, la variedad de tipos son subclases, que **heredan** métodos y atributos en común entre ellas de su clase base o modelo, la superclase. Esto se logró denominando a las clases Transporte y Paquete como “abstract” y agregando “extends (Transporte/Paquete)” en Especial, Ordinario, Automovil, Utilitario y Camion. Hubieron métodos que también fueron convertidos en abstractos cuando necesitaban ser **sobreescritos** con @Override desde las clases hijas, para cumplir dicho polimorfismo.

Cabe mencionar que en un comienzo, se debió hacer uso de “implements” para traer la firma de los métodos de la **interfaz** brindada cuya implementación es obligatoria, en EmpresaAmazing.

No resultó necesaria la **sobrecarga**, es decir, no hay métodos iguales pero con distintos parámetros. si hay, aunque no metodos mios, sino de la interfaz

Estos conceptos de programación orientada a objetos sirven para evitar la redundancia de código y por ende ahorrar memoria. Además, facilitan la lectura del diseño y aportan la posibilidad de una futura extensión de manera sencilla.

V)

//_____EmpresaAmazing_____

n= cantidad de pedidos

```
public boolean quitarPaquete(int codPaquete) {
      //n (recorre todos los pedidos)
            for (Pedido p : pedidos.values())
      //n.[quitarPedido()] (por cada pedido consulta el método)
                  if (p.quitarPaquete(codPaquete))
                        return true;
      //1     throw new RuntimeException("ERROR: el paquete no
está registrado.");
}
```

f(n) = n + n.[quitarPaquete()] + 1

f(n) = 1 + n + n.m

O(1 + n + n.m)

R2: O(1) + O(n) + O(n.m)

R2: O(max{1,n,n.m})

R1: O(n.m)

1 <= n

n <= n.m si m es mayor que cero

//_____Pedido_____

m= cantidad de paquetes

```
public boolean quitarPaquete(int codPaquete) {
      O(m) (recorre paquetes)
            for (Integer cod: carrito.keySet())
      O(m) (pregunta por cada paquete)
                  if (cod == codPaquete) {
                        carrito.remove(cod);
                        return true;
                  }
      O(1) return false;
}
```

g(n) = O(m) + O(m) + O(1)

R2: O(max{1,m})

R1: O(m)

V)

EmpresaAmazing

- transportes: no deben repetirse.

facturacion = total de pedidos cerrados

- pedidos: no debe repetirse el cliente.

Pedido

carrito:

- no deben repetirse los paquetes.

- los paquetes deben ser del tipo del tipo esperado para la clase (especial, ordinario o ambos).

- cada paquete debe tener distinto codPaquete.

Transporte

- patente: no vacia

- valorXviaje, volMax >0

Automovil

- patente: no vacia

se repite

- maxPaq>0
- valorXviaje, volMax >0

Utilitario

- patente: no vacia
- valorExtra>0
- valorXviaje, volMax >0

Camion

- patente: no vacia
- valorXviaje, volMax, porcentaje, adicional >0

Paquete

- volumen, precio >0

costoEnvio

Especial

- volumen, precio, costoEnvio >0

Ordinario

- volumen, precio, costoEnvio, porcentaje, adicional >0

Cliente

- nombre, direccion no vacíos.
- dni>=10000000.