OLIVERA NICOLÁS

COM 1

IRREP

La implementación consta de 10 clases, las cuales son “Empresa”, “Transporte”, ”TranspCSF”, “Trailer”, ”MegaTrailer”, “Flete”, “Deposito”, ”DepTercerizFrio”, “Paquete” y “Viaje”. Cada una de ellas con su correspondiente responsabilidad. Para el desarrollo del mismo, se utilizaron varios conceptos importantes, tales como lo son “Herencia”, “Polimorfismo” y “Sobreescritura”. A continuación, se analizará el IRREP de cada una de las clases:

EMPRESA

Esta clase cuenta con las variables de instancia de tipo String nombre, String CUIT, y a su vez, para almacenar las instancias pertenecientes a las subclases de Transporte y TranspCSF, Deposito y Viaje|, utilizamos como estructura de datos ArrayList. Los depósitos, al igual que los Transportes, almacenaran en otro ArryList de tipo Paquete los objetos de la clase Paquete. Respecto a las variables de tipo String podemos decir que el constructor no va a permitir ingresar cadenas vacías, de lo contrario el programa arrojará una excepción.

TRANSPORTE

En esta clase nos encontramos con las variables de instancia String numeroID, double PesoMax, double VolumenMax y double CostoPorKM. Los posibles valores de estas variables, los cuales van a ser controlados por el constructor de dicha clase, son los siguientes. String numeroID nunca va a poder ser una cadena vacía, double CostoPorKM va a poder ser mayor o igual a 0, la posibilidad de que un transporte no tenga costo por km, aunque sea muy poco probable, puede existir. Sin embargo no se permitirá ingresar un dato menor a 0. El costo final del viaje será igual a la multiplicación de del costo por kilómetro por la distancia al destino del viaje asignado. Las demás variables de tipo double, PesoMax y VolumenMax, deberán cumplir el requisito de ser estrictamente mayores a 0, ya que un transporte debe poder tener la capacidad de almacenar paquetes. Respecto a estas dos variables, podemos decir que se encuentran relacionadas con otras dos

también de tipo double llamadas pesoCargado y volCargado. Como se observa en la implementación, a medida que vamos agregando paquetes al transporte, estas cuatro variables se van actualizando. Si agregamos un paquete, el volumen y el peso máximo del transporte disminuyen de acuerdo al peso y volumen del paquete. De forma contraria, el peso y el volumen cargado aumentan. En el momento en el cual el transporte ejecuta la descarga de paquetes, para volver el volumen y el peso máximo al estado original, se toman sus valores y se los suman con los del peso y volumen ya cargados. Estas instancias de la clase Paquete serán almacenadas en un ArrayList| de tipo Paquete. Entonces, el constructor contará con cuatro parámetros uno para cada variable de instancia. En caso de que no se cumpla al menos uno de estos requerimientos, se arrojará la excepción correspondiente.

TRANSPCSF

Esta clase, cuya superclase es Transporte, es decir, que además de contar con las mismas variables de instancia, también comparte su comportamiento, también cuenta con sus propios atributos. Si no se respetan los valores válidos para esas variables de instancia, el constructor de la superclase se encargara de arrojar la excepción correspondiente. A su vez, TranspCSF es superclase de las clases Trailer y MegaTrailer. La particularidad de esta clase es que nos permitirá construir objetos que, además de tener los atributos propios de un objeto de tipo Transporte, también contará con la característica de poder tener equipo de refrigeración o no, y de tener seguro de carga o no. Si cuenta con equipo de frío, solo podrá almacenar paquetes que necesiten frío, y si no, solo se podrán cargar al transporte paquetes que no necesiten frío. Para guardar el dato, poseemos una variable de tipo boolean llamada conFrio. Si se ingresa true, es porque el transporte cuenta con equipo de refrigeración, de lo contrario se tendrá que ingresar false. Para almacenar el valor del seguro de carga, tenemos la variable de tipo double llamada seguro. Este valor deberá ser mayor o igual a 0. Si no, se arrojará excepción. Hay que agregar también que el seguro de carga se sumará al costo final de los viajes que realicen estos transportes.

TRAILER

Como se mencionó anteriormente, Trailer es subclase de TranspCSF. Es decir que comparte atributos y comportamientos, además de sus métodos. El constructor de esta clase tendrá la misma cantidad de parámetros, por lo tanto, podemos decir que su IRREP es el mismo.

MEGATRAILER

Debido a que esta clase, es subclase de TranspCSF, que a su vez es subclase de Transporte, comparte los mismos atributos y métodos de dichas clases. Sin embargo, MegaTrailer posee dos variables de instancia propias que hacen que su IRREP no sea el mismo. Ambas variables son de tipo double, una correspondiente al costo fijo (double costofijo) del transporte por viaje, y la otra para almacenar el costo extra del viaje (double costoextra). Estos datos serán agregados al valor del costo final de viaje. Los valores posibles para estas variables son datos mayores o iguales a 0. Para controlar que esto se cumpla, el programa podrá arrojar excepción en caso de ser necesario.

FLETE

Es subclase de Transporte, por ende como sucede con las clases anteriormente mencionadas, las variables de instancia serán controladas por un lado, por el constructor de la superclase. Por to lado, el constructor de esta clase también será clave para asegurar que el IRREP se cumpla.

Flete incorpora dos atributos, uno de tipo double llamado costoPorAcompañante y el otro, de tipo int llamado pasajeros. Los dos datos ingresados deberán ser mayores o iguales a 0. El valor de costoPorAcompañante por la cantidad de pasajeros será sumado al valor del costo final del viaje.

Un transporte de tipo flete, nunca podrá transportar paquetes que necesiten frio debido a que no cuenta con equipo de refrigeración.

DEPOSITO

Esta clase, al igual que Transporte, posee un atributo que determina la capacidad máxima de almacenamiento. Para guardar ese dato utilizamos una variable de tipo double llamada “capacidadMax”, la cual deberá ser estrictamente mayor a 0. Esto se debe a que un depósito tiene que contar con la capacidad de almacenar paquetes. Estas instancias de la clase Paquete serán almacenados en un ArrayList de tipo Paquete (ArrayList<Paquete>paquetes). La clase Depósito también posee otras variables de instancia como por ejemplo si cuenta o no con equipo de refrigeración, o si es propio o tercerizado. Para almacenar los correspondientes datos (true o false), contamos con las variables de tipo boolean llamada frigorífico y propio. Entonces, el constructor de la clase cuenta con tres parámetros, uno para ingresar la capacidad máxima, y otros dos para ingresar datos de tipo boolean (true o false) que se guardaran en

las variables anteriormente nombradas. Si la capacidad máxima del depósito que se ingrese es menor o igual a 0 se arrojará una excepción.

Hay que agregar también que si un depósito cuenta con frigorífico, solo podrá almacenar paquetes que necesiten frío, y de forma contraria, si no cuenta con equipo de refrigeración solo podrá almacenar paquetes que no necesiten frío.

DEPTERCERIZFRIO

Posee a Depósito como superclase, por ende comparte mismos atributos y métodos. Sin embargo, podemos decir que la cantidad de parámetros del constructor no es la misma, pero además, estos depósitos poseen un determinado costo por tonelada cargada. Debido a que estamos hablando de un depósito tercerizado con equipo de refrigeración, tenemos que obviar los valores de las variables confrio y propio pertenecientes al constructor de la superclase. Estos van a ser igual a true y false respectivamente por defecto. El constructor de esta clase solo contara con dos parámetros de tipo double. Uno para ingresar la capacidad máxima, la cual debe ser mayor a 0, y otro para ingresar el costo por tonelada cargada. Este deberá ser mayor o igual a 0. El valor de la capacidad máxima va a ser "controlado" por el constructor de la superclase Deposito (valor mayor a 0), pero el costo por tonelada será controlado por el constructor de esta clase.

PAQUETE

Esta clase cuenta con cuatro variables de instancia, por ende nuestro constructor contará con esa cantidad de parámetros. Un paquete posee un peso, un volumen, un destino y puede necesitar refrigeración o no. Se utilizaron variables de tipo doublé tanto para el peso como para el volumen, boolean para guardar el dato si necesita frio o no (true, false) y para el destino tenemos una variable de tipo String, Respecto al peso y al volumen, estos deben ser mayores o iguales a 0 y el destino no podrá ser una cadena vacía. Caso contrario, se arrojará excepción.

VIAJE

Los viajes cuentan solo con dos atributos, un destino y una distancia. Es por eso que para esta clase utilizamos solo dos variables de instancia. Una variable de tipo int para la distancia (kilómetros) y otra de tipo String para el nombre del destino. Respecto a los datos válidos para estas variables, podemos decir que la distancia no puede ser menor a 0, y el destino no puede ser una cadena vacía. Estos dos casos van a ser controlados por el constructor de la clase. Si no se respetan se arrojara excepción.

Método equals de la clase Transporte:

```
public boolean equals(Object obj) {                    (Cantidad de operaciones)
        if (this == obj)                                                    2
                return true;                                                1
        if (obj == null)                                                    2
                return false;                                               1
        if (getClass() != obj.getClass())                                   4
                return false;                                               1
        Transporte other = (Transporte) obj;                                2
        if (Double.doubleToLongBits(Volcargado) !=                          4
    Double.doubleToLongBits(other.Volcargado))                              2
                return false;                                               1
        if (destino == null) {                                              2
                if (other.destino != null)                                  3
                        return false;                                       1
        } else if (!destino.equals(other.destino))                          4
                return false;                                               1
```

```
            if (Double.doubleToLongBits(pesoCargado) !=        3
Double.doubleToLongBits(other.pesoCargado))|                   2
                return false;                                  1
            return true;                                       1
        }                                          Total=  38 op
```

*Orden de omplejidad del algoritmo = O(1)