Programación II - Trabajo Práctico Integrador
2do Cuatrimestre 2020
SEGUNDA PARTE

Fecha de presentación: martes 26 de mayo
Fecha de entrega: martes 9 de junio

En esta segunda parte deben entregar la implementación, análisis de complejidad en donde se pida, y el IREP para la representación de datos seleccionada.

Requerimientos técnicos:

- Grupos de 1 ó 2 personas.

- Se deben utilizar las herramientas de Tecnologías Java que se vieron en la materia. Al menos una vez deben usarse:

  - *Stringbuilder,* cuyo uso debe basarse en la necesidad de modificar el string.
  - *Iteradores y Foreach* para recorrer las colecciones de Java

Se utilizará en el desarrollo del trabajo al menos 3 de estos conceptos: **herencia, polimorfismo, sobreescritura, sobrecarga e interfaces**. Como también, en los casos que corresponda, se deberá implementar clases abstractas.

Por otro lado, desde la materia se proveerá un código cliente y la explicación de sus métodos para que se utilicen como base para la implementación, como también un test junit. Será condición necesaria para esta presentación que tanto el código cliente como el test ejecuten sin errores.

- Además de pasar el test de *junit* suministrado junto con el TP, en la corrección se testean los ejercicios con otro junit adicional, por lo que se recomienda armar un conjunto propio de testeo acorde a su implementación, antes de entregar el TP.
- **Escribir el *IREP* de la representación elegida para la implementación**. Se debe entregar por escrito.

Consideraciones importantes para la implementación:

La implementación de los TADs deben responder al diseño presentado en la Primera Parte *teniendo en cuenta las correcciones que se indicaron a cada grupo*.

- Deberá correr satisfactoriamente con el código cliente entregado
- Deberá pasar satisfactoriamente el test junit proporcionado.
- El código debe tener implementado el método ***toString*** de la Empresa,
- Se deberá implementar un método ***equals*** que responda al siguiente requerimiento:

La Empresa necesita que, dada la Identificación de un Transporte, se devuelva la identificación de otro Transporte igual (el primero que se encuentre). Se considera que dos transportes son iguales si : - Son el mismo tipo de transporte, - tienen el mismo destino y - llevan la misma carga de paquetes. Retornar null si no existe.

String obtenerTransporteIgual(String idTransp)

- Explicar cuál es la complejidad del método equals anterior. **Entregar por escrito**.

Test1: De debe testear al menos el siguiente código además del junit que se subirá el jueves 28

```
public static void main(String[] args) {
    double cargado;
    Empresa e = new Empresa("30112223334","La Santafesina");
    System.out.println(e.toString());
    e.agregarDepTercerizFrio(40000, 10);
    e.agregarDeposito(50000, true, true);
    e.agregarDeposito(80000, false, true);
    e.agregarDeposito(90000, false, false);
    e.agregarDestino("Rosario", 100);
    e.agregarDestino("Buenos Aires", 400);
    e.agregarDestino("Mar del Plata", 800);
    e.agregarTrailer("AA333XQ", 10000, 60, true, 2, 100);
    e.agregarMegaTrailer("AA444PR", 15000, 100, false, 3, 150, 200, 50);
    e.agregarFlete("AB555MN", 5000, 20, 4, 2, 300);
    e.asignarDestino("AA333XQ", "Buenos Aires");
    e.asignarDestino("AB555MN", "Rosario");
    // paquetes que necesitan frio
    e.incorporarPaquete("Buenos Aires", 100, 2, true);
    e.incorporarPaquete("Buenos Aires", 150, 1, true);
```

```
        e.incorporarPaquete("Mar del Plata", 100, 2, true);
        e.incorporarPaquete("Mar del Plata", 150, 1, true);
        e.incorporarPaquete("Rosario", 100, 2, true);
        e.incorporarPaquete("Rosario", 150, 1, true);
        // paquetes que NO necesitan frio
        e.incorporarPaquete("Buenos Aires", 200, 3, false);
        e.incorporarPaquete("Buenos Aires", 400, 4, false);
        e.incorporarPaquete("Mar del Plata", 200, 3, false);
        e.incorporarPaquete("Rosario", 80, 2, false);
        e.incorporarPaquete("Rosario", 250, 2, false);
        cargado = e.cargarTransporte("AA333XQ");
        System.out.println("Se cargaron "+cargado+" metros cubicos en el transp
                                    AA333XQ");
        e.iniciarViaje("AA333XQ");
        System.out.println("Costo del viaje: "+e.obtenerCostoViaje("AA333XQ"));
        System.out.println(e.toString());
        e.finalizarViaje("AA333XQ");
        System.out.println(e.toString());
}
```

Donde se utilizan los siguientes métodos

- **int agregarDeposito(double capacidad, boolean frigorifico, boolean propio)** que agrega un depósito a la empresa. Retorna un número de depósito, que debe ser único dentro de la empresa.
- **int agregarDepTercerizFrio(double capacidad, double costoPorTonelada)** que agrega a la empresa un depósito tercerizado frigorífico. Retorna un número de depósito, que debe ser único dentro de la empresa.
- **void agregarDestino(String destino, int km)** Incorpora un nuevo destino y su distancia en km. Es requisito previo, para poder asignar un destino a un transporte.

Los siguientes métodos agregan los tres tipos de transportes a la empresa, cada uno con sus atributos correspondientes:

- **void agregarTrailer(String idTransp, double cargaMax, double capacidad, boolean frigorifico, double costoKm, double segCarga)**
- **void agregarMegaTrailer(String idTransp, double cargaMax, double capacidad, boolean frigorifico, double costoKm, double segCarga, double costoFijo, double comida)**
- **void agregarFlete(String idTransp, double cargaMax, double capacidad, double costoKm, int acomp, double costoPorAcom)**

Se asigna un destino a un transporte dado su ID (el destino debe haber sido agregado previamente, con el método agregarDestino):

- **void asignarDestino(String idTransp, String destino)**

Se incorpora un paquete a algún depósito de la empresa. Devuelve verdadero si pudo incorporarlo, es decir, si encontró un depósito acorde al paquete y con espacio suficiente:

- **boolean incorporarPaquete(String destino, double peso, double volumen, boolean frio)**

Dado un ID de un transporte se pide cargarlo con toda la mercadería posible, de acuerdo al destino del transporte. No se debe permitir la carga si está en viaje o si no tiene asignado un destino. Utiliza los depósitos acordes para cargarlo. Devuelve un double con el volumen cargado en el transporte:

- **double cargarTransporte(String idTransp)**

Inicia el viaje el transporte cuyo ID se pasa por parámetro. No se debe permitir iniciar el viaje sin mercadería cargada:

- **void iniciarViaje(String idTransp)**

Finaliza el viaje del transporte cuyo ID se pasa por parámetro. El transporte vacía su carga y blanquea su destino, para poder ser vuelto a utilizar en otro viaje:

- **void finalizarViaje(String idTransp)**

Obtiene el costo de viaje del transporte cuyo ID se pasa por parámetro:

- **double obtenerCostoViaje(String idTransp)**