Programación II - Trabajo Práctico Integrador
1er Cuatrimestre 2022
SEGUNDA PARTE

Fecha de presentación: jueves 19 de mayo
Fecha de entrega: jueves 2 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.
Para poder empezar con la segunda parte deben tener aprobado el diseño presentado en la primera parte.

Requerimientos técnicos:

- Grupos de 1 ó 2 personas (**El mismo grupo de la primera parte del TP)**.

- Se deben utilizar donde sea conveniente 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 deberá utilizar 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**. En el informe se debe explicar qué conceptos se utilizaron, dónde y de qué modo.

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 una clase de testeo (junit). Será condición necesaria para esta presentación que tanto el código cliente como el test se 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 debe responder a su 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.
- Deberá aprovechar correctamente las estructuras de datos elegidas.
- 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 matrícula de un Transporte, se devuelva la matrícula 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 matricula)
  ```

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

Test (JUnit):

Se habilitará el archivo de test en el moodle, junto a este enunciado, de donde deberán descargarlo.

Código Cliente:

```
public static void main(String[] args) {
      double volumen;
      Empresa e = new Empresa("30112223334","Expreso Libre”, 40000);
      System.out.println(e.toString());
      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);
        volumen = e.cargarTransporte("AA333XQ");
        System.out.println("Se cargaron " + volumen
                           +" 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());
}
```

Se debe respetar la siguiente Interfaz:

```
// Constructor de la empresa.
Empresa(String cuit, String nombre, int capacidadDeCadaDeposito);

// Incorpora un nuevo destino y su distancia en km.
// Es requisito previo, para poder asignar un destino a un transporte.
// Si ya existe el destino se debe generar una excepción.
void agregarDestino(String destino, int km);

// Los siguientes métodos agregan los tres tipos de transportes a la
// empresa, cada uno con sus atributos correspondientes.
// La matrícula funciona como identificador del transporte.
void agregarTrailer(String matricula, double cargaMax,
                    double capacidad, boolean tieneRefrigeracion,
                    double costoKm, double segCarga);
void agregarMegaTrailer(String matricula, double cargaMax,
                    double capacidad, boolean tieneRefrigeracion,
                    double costoKm, double segCarga, double costoFijo,
                    double costoComida);
void agregarFlete(String matricula, double cargaMax, double capacidad,
                  double costoKm, int cantAcompaniantes,
                  double costoPorAcompaniante);

// Se asigna un destino a un transporte dada su matrícula (el destino
// debe haber sido agregado previamente, con el método agregarDestino).
// Si el destino no está registrado, se debe generar una excepción.
void asignarDestino(String matricula, String destino);

// Se incorpora un paquete a algún depósito de la empresa.
```

```
// Devuelve verdadero si se pudo incorporar, es decir,
// si el depósito acorde al paquete tiene suficiente espacio disponible.
boolean incorporarPaquete(String destino, double peso, double volumen,
                          boolean necesitaRefrigeracion);

// 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.
// Utilizar el depósito acorde para cargarlo.
// Devuelve un double con el volumen de los paquetes subidos
// al transporte.
double cargarTransporte(String matricula);

// Inicia el viaje del transporte identificado por la
// matrícula pasada por parámetro.
// En caso de no tener mercadería cargada o de ya estar en viaje
// se genera una excepción.
void iniciarViaje(String matricula);

// Finaliza el viaje del transporte identificado por la
// matrícula pasada por parámetro.
// El transporte vacía su carga y blanquea su destino, para poder
// ser vuelto a utilizar en otro viaje.
// Genera excepción si no está actualmente en viaje.
void finalizarViaje(String matricula);

// Obtiene el costo de viaje del transporte identificado por la
// matrícula pasada por parámetro.
// Genera Excepción si el transporte no está en viaje.
double obtenerCostoViaje(String matricula);

// Busca si hay algún transporte igual en tipo, destino y carga.
// En caso de que no se encuentre ninguno, se debe devolver null.
String obtenerTransporteIgual(String matricula);
```