Universidad Internacional de Valencia

21GIIN - PROYECTOS DE PROGRAMACIÓN

Elementos POO

GRUPO 11

Luis Valbuena Arribas

Benjamín Miguel Miralpeix

De:
Planeta Formación y Universidades

INTRODUCCIÓN

Se plantea realizar un proyecto de gestión logística de una empresa portuaria, en dónde se deberán gestionar las rutas, empaquetado y facturación de la mercancía siguiendo la documentación con los requerimientos proporcionados para esta primera entrega:

- Herramientas a utilizar.
- Estructura básica del proyecto, organización de clases, métodos, elementos de POO, etc.
- División y organización de tareas.

Al final de este documento se proporcionan los diferentes enlaces de las herramientas utilizadas como GitHub en dónde se encuentra almacenado el proyecto (diagramas, código, etc.), Trello en dónde se organiza e informa de las tareas realizadas.

HERRAMIENTAS UTILIZADAS

GitHub

Debido a la estabilidad que proporciona, facilidad de manejo para el control de versiones y a la posibilidad de que cada miembro del equipo pueda trabajar en ramas separadas se ha creado un repositorio para almacenar todo lo relacionado con el proyecto. Una vez finalizado el proyecto se unificarán las ramas para la entrega final.

Trello

Nos va a permitir gestionar el proyecto y el flujo de trabajo, así como supervisar las tareas y verificar que se cumplen los requisitos de cada actividad.

PlantUML/Flowdia

Nos va a permitir crear los diagramas de clases y los casos de uso.

GoogleMeets

Es la herramienta elegida para realizar las reuniones de los integrantes del equipo. Estas reuniones se plantearán a modo de *dailys* en dónde se comentarán las posibles dudas, bloqueos, avances, etc. Las reuniones están planificadas una vez a la semana, aunque siempre se intentará aumentar la cantidad de reuniones a la semana.

WhatsApp

Uso de esta herramienta para una comunicación más directa y diaria.

Visual Studio Code

Hemos elegido el uso de *Visual Studio Code* debido a la facilidad para trabajar con este *IDE*. La otra opción que teníamos era *NetBeans* pero debido al poco uso de la herramienta pensamos que podría ralentizarnos en nuestro trabajo.

DIVISIÓN DE TAREAS

En las primeras reuniones se ha desarrollado conjuntamente el tema de clases, métodos, etc. Para las siguientes entregas en las que tengamos que diseñar los casos de uso, diagrama de clases, etc., también lo haremos conjuntamente, ya que es complicado que cada uno diseñe los diagramas por su cuenta. Creemos que es mejor juntarnos entre los dos e ir avanzando en la misma dirección.

En posteriores tareas como por ejemplo la implementación de código concreto, la división de tareas se discutirá en reuniones en las que se plantearán los aspectos más importantes sobre la tarea, así como su estructura y la división del trabajo. La división se hará a través del tablero de Trello, asignando de forma equitativa la carga de la tarea. No obstante, si detectamos bloqueos o errores, nos juntaremos para intentar realizar un desarrollo conjunto de una clase o método en particular.

ORGANIZACIÓN DE CLASES – ELEMENTOS POO

1. **Registro al sistema:**
    - Clase **Usuario** con atributos como nombre, contraseña, rol, etc.
    - Clase **Rol** para gestionar los permisos asociados a cada rol.
    - Métodos para crear, leer, actualizar y eliminar usuarios (CRUD).
2. **Operaciones:**
    - Clase **Operacion** con atributos que relacionen la información de ruta, empaquetado y facturación.
    - Métodos para manejar las operaciones, como crear una nueva operación, actualizar sus detalles, eliminar una operación, etc.
3. **Rutas:**
    - Clases específicas para cada tipo de ruta: **RutaTerrestre**, **RutaMaritima**, **RutaAerea**, cada una con sus atributos específicos.
    - Métodos para gestionar la creación y manejo de rutas.
4. **Empaquetado:**
    - Clase **Mercancia** o Paquete con atributos como número de contenedores, peso, tamaño, etc.
    - Métodos para gestionar la información relacionada con la mercancía.
5. **Facturación:**
    - Clase **Factura** con métodos para generar facturas a partir de las líneas pendientes a facturar, validar datos y mantener un registro de facturas generadas.

CLASES Y MÉTODOS

A continuación, vamos a comentar las clases, atributos, métodos, etc. que hemos creado en esta primera entrega para la realización del proyecto.

public class Operacion

Dentro de la clase operación se incluyen una serie de atributos de tipo **private** como tipo, cantidad e id para definir el tipo de operación, la cantidad de contenedores y el identificador de la operación respectivamente. La clase operación va a estar relacionada con las clases ruta, empaquetado y facturación a través de los atributos ruta, empaquetado y facturación.

Al tener atributos como por ejemplo **private Ruta ruta**, lo que se quiere es encapsular la información relacionada con la ruta dentro de cada objeto Operación, permitiendo que la clase Operación tenga su propia referencia a un objeto Ruta. Esto nos ayudará a organizar y estructurar la información de manera más eficiente y coherente dentro de cada instancia de la clase Operación

Aquí analizamos paso a paso la clase Operación:

1. **Atributos:** La clase **Operacion** tiene atributos que representan detalles específicos de una operación logística: **tipo, cantidad, id, ruta, empaquetado y facturacion**. Estos atributos almacenan información sobre el tipo de operación, la cantidad de contenedores, un identificador único, la ruta asociada, los detalles de empaquetado y los detalles de facturación.
2. **Constructor:** El constructor **Operacion** se utiliza para crear objetos de tipo **Operacion**. Toma como parámetros los valores necesarios para inicializar los atributos de una operación específica: tipo, cantidad, ruta, empaquetado y facturación.
3. **Getters y Setters:** Se han definido métodos **get** para obtener los valores de los atributos y métodos **set** para modificar o asignar valores a

los atributos específicos de una instancia de **Operacion**. Por ejemplo, **getTipo()** devuelve el tipo de operación, y **setTipo()** permite cambiar el tipo de operación.

4. **Métodos de operaciones:** Se han incluido métodos (**crearOperacion(), eliminarOperacion(), modificarOperacion()**) para realizar operaciones específicas relacionadas con las operaciones logísticas. Sin embargo, estos métodos están actualmente vacíos y podrían ser implementados según las necesidades específicas de la aplicación.
5. **Método toString:** Este método sobrescrito permite obtener una representación en formato de cadena de texto de los atributos de una instancia de **Operacion**.

public class Ruta

Se crea la clase Ruta que va a ser utilizada para incluir los datos de las diferentes rutas de transporte. Entre los atributos de la clase Ruta se incluyen datos como el tipo de ruta, distancia, origen, destino, coste, etc. necesarios para tener la información lo más detallada posible de una ruta.

A continuación, se detalla cada parte del código:

1. **Atributos:** La clase Ruta tiene siete atributos: **tipoRuta**, **puertoOrigen**, **puertoDestino**, **distancia**, **duracion**, **coste**, **id**. Estos atributos representan diferentes características de una ruta, como el tipo de ruta, los puertos de origen y destino, la distancia en kilómetros, la duración en horas, el costo y un identificador único.
2. **Constructor:** El constructor Ruta se utiliza para crear objetos de tipo Ruta. Toma como parámetros los valores necesarios para inicializar los atributos de una ruta específica: tipo de ruta, puertos de origen y destino, distancia, duración, costo e identificador.
3. **Getters y Setters:** Se han definido métodos get para obtener los valores de los atributos y métodos set para modificar o asignar valores a los atributos específicos de una instancia de Ruta. Por ejemplo,

getTipoRuta() devuelve el tipo de ruta, y **setTipoRuta()** permite cambiar el tipo de ruta.

4. **Métodos:**

- **calcularTiempoEstimadoLlegada(double velocidadPromedio):** Este método toma como parámetro una velocidad promedio en kilómetros por hora y calcula el tiempo estimado de llegada para recorrer la distancia de la ruta. Utiliza la fórmula básica de distancia = velocidad * tiempo para calcular el tiempo necesario para recorrer la distancia basándose en la velocidad proporcionada.
- **calcularCostoTotalEstimado(int costoPorKilometro):** Este método calcula el costo total estimado de la ruta basándose en el costo por kilómetro dado como parámetro. Multiplica la distancia de la ruta por el costo por kilómetro para obtener el costo total estimado.

Estos métodos añaden funcionalidad adicional a la clase Ruta, permitiendo cálculos relacionados con el tiempo estimado de llegada y el costo total estimado, lo cual puede ser útil para cálculos y estimaciones relacionadas con la logística y planificación de las rutas.

Estos elementos en conjunto permiten crear, almacenar y acceder a información sobre una ruta en particular dentro de tu aplicación. Los **getters** y **setters** ofrecen la interfaz para acceder y manipular los datos de una instancia de la clase Ruta.

Al igual que en el caso anterior, los atributos y métodos creados en esta clase pueden ser modificados en un futuro debido a las exigencias del proyecto.

public class Empaquetado

1. **Atributos:** La clase Empaquetado tiene tres atributos que representan características del empaquetado: **numeroContenedores, peso y tamano**. Estos atributos almacenan información sobre la cantidad de contenedores, el peso total y el tamaño del empaquetado.

2. **Constructor:** El constructor **Empaquetado** permite crear objetos de tipo **Empaquetado**. Toma como parámetros el número de contenedores, el peso y el tamaño del empaquetado, y los asigna a los atributos correspondientes de la instancia creada.
3. **Getters y Setters:** Se han agregado métodos **get** y **set** para los atributos de la clase Empaquetado. Estos métodos permiten acceder y modificar los valores de los atributos de manera controlada desde fuera de la clase.
4. **Método calcularDensidad:** Este método calcula la densidad del empaquetado dividiendo el peso entre el tamaño del empaquetado. Devuelve un valor que representa la densidad del contenido del empaquetado, lo cual puede ser útil en el contexto logístico para evaluar la relación entre el peso y el tamaño ocupado.

Se han añadido algunos métodos para ofrecer más funcionalidad a esta clase y así poder obtener información sobre el empaquetado y manipular sus atributos.

public class Facturacion

1. **Atributos:** La clase **Facturacion** tiene atributos que representan información asociada a una factura, como el **código, fecha, hora, tipo, matrícula, nombre, DNI, concepto, importe y estado**.
2. **Constructor:** El constructor **Facturacion** permite crear objetos de tipo **Facturacion** al proporcionar valores para todos los atributos de la clase.
3. **Getters y Setters:** Se han definido métodos **get** y **set** para cada atributo. Estos métodos permiten obtener y modificar los valores de los atributos de manera controlada desde fuera de la clase.

    - **Getters (get):**
        1. **getCodigo(), getFecha(), getHora(), getTipo(), getMatricula(), getNombre(), getDni(), getConcepto(), getImporte(), getEstado():** Estos métodos permiten obtener el valor de los atributos correspondientes de la factura. Por

ejemplo, **getCodigo()** devuelve el código de la factura, **getFecha()** devuelve la fecha asociada y así sucesivamente para los demás atributos.

- **Setters (set):**
    1. **setCodigo(String codigo), setFecha(String fecha), setHora(String hora), setTipo(String tipo), setMatricula(String matricula), setNombre(String nombre), setDni(String dni), setConcepto(String concepto), setImporte(double importe), setEstado(String estado):** Estos métodos permiten asignar un valor nuevo a los atributos correspondientes de la factura. Por ejemplo, **setCodigo(String codigo)** establece un nuevo código para la factura, **setFecha(String fecha)** cambia la fecha, y así sucesivamente para los demás atributos.

Estos métodos **get** y **set** aseguran que los datos de la factura se accedan y modifiquen de manera controlada, manteniendo la encapsulación y la integridad de los datos dentro de la clase **Facturacion**.

En próximas entregas tenemos pensado implementar interfaces, herencia, clases abstractas, etc., ya que ahora mismo el proyecto está en una fase muy “verde” y todavía necesitamos desarrollar más en profundidad el diagrama de clases y los casos de uso para tener una visión mucho más completa de cómo queremos que sea nuestro proyecto definitivo.

Enlaces: https://github.com/BenjaLuke/ProyectoProgramacion