

Documentación del Proyecto

Miembros del grupo

Miguel Moretón, Pelayo Castro, Miguel Santos, Marco Benali

Introducción

El proyecto consiste en un programa que simula la aparición aleatoria de pedidos y su reparto. El programa cuenta con las siguientes clases:

1. Clientes: son los que realizan los pedidos y tienen asignada una ciudad entre otros atributos. Hay un total de 30 clientes.
2. Empleados: estos pueden ser conductores o repartidores. Hay 3 de cada tipo.
3. Ciudades: se representan mediante coordenadas en un plano de 2 dimensiones. Hay un total de 20 ciudades y una de ellas tiene asignado el rol de almacén, desde el que se reparten los productos a las otras ciudades.
4. Productos: son pedidos por los clientes asignados a ciudades diferentes. Los productos tienen una disponibilidad, precio y cantidad como atributos principales.
5. Ruta: es una secuencia de índices de ciudades. La ruta debe ser aquella que minimice la distancia total recorrida y que visite todas las ciudades empezando y terminando en el almacén.
6. Pedido: van asociados a un cliente y contienen una canasta aleatoria de productos que se generan automáticamente cuando se ejecuta el programa.

Enlace al Repositorio

<https://github.com/PelayoCastro/Proyecto-de-Programaci-n>

Estructura del Proyecto: Diagrama de Clases

A continuación, la representación de las clases en pseudocódigo, mostrando los atributos y métodos públicos y privados.

```
abstract class Persona {  
    -string nombre  
    -string apellido1  
    -string apellido2  
    -Fecha fechaNacimiento  
    -long long numeroTelefonico  
    -Ciudad ciudad  
    +mostrar() void  
}
```

```
class Cliente {  
    -string clienteID  
    +mostrar() void
```

```
+getID() string  
}
```

```
class Empleado {  
    -string empleadoID  
    -Fecha fechaIncorporacion  
    -Tipo tipo  
    +mostrar() void  
    +getID() string  
    +getTipo() Tipo  
}
```

```
class Ciudad {  
    -Coordenadas coordenadas  
    -bool esAlmacen  
    -float distanciaAlmacen  
    +getCoordenadas() Coordenadas  
    +getEsAlmacen() bool  
    +getDistanciaAlmacen() float  
}
```

```
class Producto {  
    -string nombreProducto  
    -int cantidadDisponible  
    -float precio  
    +getNombre() string  
    +getCantidad() int  
    +getPrecio() float  
}
```

Relaciones entre clases

Persona \leftarrow Cliente (Cliente (subclase) hereda de Persona (superclase))

Persona \leftarrow Empleado (Empleado (subclase) hereda de Persona (superclase))

Persona – Ciudad (La clase Persona está asociada a la clase Ciudad)

Cuando se ejecuta el código se crean nuevas relaciones:

Pedido – Cliente (Un cliente realiza pedidos)

Pedido – Producto (Un pedido consta de varios productos)

Ruta – Ciudad (Una ruta consta de un conjunto de ciudades)









Metodología Agile

La metodología Agile es una filosofía de creación de software en la que se trabaja con *sprints* o iteraciones incrementales. A continuación, los detalles de cada iteración:

Sprint 1: Estructura de Clases

En un primer lugar creamos las clases que vamos a necesitar (Clientes, Empleados, Productos, Ciudades, Ruta, Pedidos) así como la clase abstracta Persona. En segundo lugar, creamos los ficheros para las clases Clientes, Empleados, Productos y Ciudades. El resto de las clases no necesitarán ficheros.

Clases: Persona {abstracta}, Cliente, Empleado, Producto, Pedido, Ruta, Ciudad.

Clases	Archivo de Texto ( → necesario  → no necesario
Clientes	
Empleados	
Productos	
Ciudades	
Ruta	
Pedidos	

Sprint 2: Gestión de Archivos

En el segundo sprint creamos los métodos que leen la información de los archivos y asignan los valores leídos a las clases. En esta etapa también se crearon métodos para el manejo de excepciones.

Sprint 3: Prueba del código

Una vez terminamos de implementar los métodos para la lectura de archivos añadimos código a la función principal que llama a los métodos creados en el sprint anterior y otro código para comprobar que los métodos funcionan correctamente.

Sprint 4: Optimización del código y documentación.

Una vez comprobamos que el código funciona correctamente procedimos a separar el código en headers y la función principal. Asimismo, también creamos la documentación asociada al código.

Funcionalidades Principales

1. Gestión de las Ciudades

Las ciudades son los puntos desde o hacia los cuales se llevan los pedidos. Hay un número fijo de ciudades (20) y una de ellas es un almacén. Primero se cargan los datos de las ciudades que se encuentran en un fichero de texto. En segundo lugar, se llaman a métodos que calculan la distancia entre las ciudades y se determina si son o no almacenes.

2. Gestión del Personal

Se cargan los datos de los empleados, los cuales pueden ser de dos tipos: conductores y repartidores. En total hay 6 empleados (3 conductores y 3 repartidores).

3. Gestión de los Clientes

Se cargan los datos de los clientes (un total de 30) y se asigna a cada cliente una ciudad de acuerdo con la información del fichero.

4. Gestión del Inventario

Se carga la lista de productos desde el fichero determinando disponibilidad, cantidad y precio. Se calcula el valor total del inventario.

Beneficios Empresariales

- El programa calcula la ruta óptima entre ciudades lo que minimiza los costes de transporte.
- El programa sigue el estado del inventario en todo momento, lo cual permite anticiparse si un producto se agota.

Estructura del Código

El programa consta de los siguientes archivos:

1. `ProyectosProgramacionFuncionPrincipal.cpp`
 - Contiene la función principal
 - Implementa la lógica principal del programa
 - Llama a los métodos de lectura de datos para asignar la información de los ficheros a las variables.
 - Muestra la información por pantalla.
2. `ProyectosProgramacionHeader.h`
 - Contiene todas las clases
 - Contiene la implementación de los métodos
 - Define las constantes y tipos de dato enumerados

Esta organización del código cumple los principios de modularidad, ya que se separa la declaración de la implementación haciendo uso de headers. Los métodos de lectura de archivos manejan excepciones en la lectura de los

ficheros, validan los datos y proporcionan mensajes de error descriptivos. Se protegen los datos declarándolos como privados y usando getters y setters cuando es necesario.

Conclusiones y Trabajo Futuro

El programa es una primera aproximación a problemas logísticos que podemos encontrarnos en el mundo real. En el futuro podría mejorarse sustituyendo la representación simplificada de las ciudades como puntos en un plano bidimensional por datos reales y algoritmos para optimizar muchas más variables como el tráfico. También podría añadirse una interfaz gráfica que represente de manera más sencilla la información mostrada en terminal. Por último, se podría crear un programa que realice predicciones de pedidos para prepararse con antelación.