UTN SANTA FE

Diseño e Implementación

De Estructuras de Datos

Guía 05 – Programación Orientada a Objetos en Java

Contenido

Guía 05 – Programación Orientada a Objetos en Java

Se propone que en Eclipse (o el IDE que utilice) crear un proyecto java y resuelva cada problema en un paquete separado.

Problema 01 - Sobrecarga

Implementar una clase Camino, que guarda una lista de Coordenadas que determinan el camino que debe seguir un repartidor para entregar todos los productos que debe repartir.

La clase Coordenadas solo posee dos atributos de tipo double, latitud y longitud.

La clase Camino, posee una lista de coordenadas, donde se supone que la primer coordenada es donde comienza el camino y la última es donde finaliza.

a. La clase Coordenada.
b. Sobrecargar los constructores. Esta clase debe ofrecer el constructor por defecto y un constructor con la latitud y la longitud
c. Agregar a la clase Coordenadas el método equals para determinar si dos coordenadas son iguales.
d. Agregar a la clase Coordenadas el método toString()
e. Agregar la clase Camino que tiene una lista de caminos (arreglo o ArrayList),
f. Crear en la clase camino los siguientes métodos "agregar"
    1. **public void agregar(Coordenada x)** : agrega un punto a la lista de destinos representado por la coordenda

2. **public void agregar(double lat, double lng)** : agrega un punto a la lista de destinos representados por la coordenada de latitud lat y longitud lng
3. **public void agregar(int mtsLt,int mtsLn)**: agrega un nuevo punto a la lista de destinos, cuya latitud y longitud será la suma de los metros recibidos como parámetros (para calcular como sumar "metros" a una latitud y longitud puede guiarse por este enlace https://stackoverflow.com/a/7478827/2370742)

g. Agregar a la clase camino el método "public List<Coordenada> buscar(Coordenada no,Coordenada se)" que retorna todos los puntos del camino que se encuentran dentro del limite establecido por la coordenada no (noroeste, es decir el límite superior izquierdo) y la coordenada se (sureste, es decir el limite inferior derecho).
h. Sobrecargar el método buscar para que reciba como primer argumento una coordenada y como segundo argumento una distancia en metros, y retorne todos los puntos que se encuentran en el radio.
    1. *Sugerencia puede usar el método del punto f inciso 3 para calcular los límites NO y SE (simplemente sumar metros en latitud y restar en longitud para obtener el limite NO, y restar metros en latitdud y sumarlos en longitud para calcular el límite SE) y luego invocar al método del inciso g.*
i. Escribir una clase App para probar todos los métodos escritos hasta aquí.

Problema 2 – Herencia y Sobrescritura

Una app de delivery de productos permite que el usuario solicite productos para el envío. Esta aplicación tiene posee dos opciones de envío, básico y premium.

Cuando un usuario selecciona un envío básico puede agregar hasta 5 productos en el envío y se le cobra además del precio de cada producto, un 5% extra del precio de cada producto. Es decir si pido un producto por $100, el costo del Pedido es $105 ($100 del producto y $5 del envío) y el envío se realiza entre las 24 y las 48 horas hábiles. El usuario tiene la opción de indicar que el envío básico es **Express,** y necesita recibirlo en 24 horas. Si lo marca como **Express** entonces se le cobra un 20% sobre el total del envío. (en el caso anterior a los $105 se le suma un 20%, es decir $21 y el costo del envío será $126).

Los envíos premium, el cliente tiene la garantía de recibir el pedido en menos de 12 horas, pero se le cobra un 20% extra sobre cada producto hasta 5 productos, y un 30% extra si son más de 5 productos los pedidos hasta un máximo de 10.

Cada pedido es asignado a un cadete, que los tiene en su lista de pedidos para repartir. El cadete cobrará por cada pedido básico el 10% del costo final y por cada pedido premium el 15% del costo final y $50 extras si el pedido tiene más de 10 productos.

Al final del día cada cadete puede calcular cuanto le corresponde en comisiones y la app se lo acredita.

Crear las siguientes clases:

- Producto: tendrá una descripción y un costo.
- Pedido y sus clases hijas, PedidoBasico y PedidoPremium.
- Agregar los atributos que considere necesario.
- Agregar a Pedido el método concreto y **final** "boolean
- Agregar a pedido básico la posibilidad de marcar el envio como "express"
- agregarProducto(Producto p)" que recibe como argumento un producto y lo agrega al pedido si es que todavía hay espacio y retorna true. Si ya alcanzó el limite máximo de productos no lo agrega y retorna false.
- Agregar en Pedido los métodos abstractos "precio()" y "comisión()" que permiten calcular para cada pedido el precio del mismo y la comisión del mismo e implementarlos en las clases concretas.
- Crear la clase Cadete, e implementar los métodos
    - "agregarPedido": agrega un pedido que tiene que llevar el cadete
    - "comisiones" que retorna el total de comisiones que cobrará el cadete por todos los pedidos que le han agregado.
- Probar la aplicación.
- Modificar la clase pedido para que los mismos tenga un campo de tipo LocalDate (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/LocalDate.html) que indica la fecha en que se entregó.
- Modificar los métodos "comisión" de la clase pedido, para que retornen una comisión a pagar solo si el pedido se entregó (es decir la fecha **no es null**). De esta manera al cadete solo se le paga por los envíos realizdos.
- ¿Fue necesario modiicar la clase cadete y el método de cálculo de las comisiones?

Problema 3 – Interfaces

La app del problema 2, ha expandido su negocio y ahora permite que los cadetes también reciban pedidos para realizar trámites (en un banco, en una oficina gubernamental, pagos en negocios como pagofácil, retirar ecomiendas y cualquier otro trámite que pueda delegarse). Por cada trámite,**realizado,** un la app le cobra al usuario un monto fijo de $50. Y un cadete por cada trámite cobra $20.

Para esto deberemos agregar una interface que permita agrupar en una misma jerarquía de elementos Comisioables, a todos los elementos a los cuales

se les puede calcular la comisión a pagar al Cadete. Esta interface será implementada por los pedidos, y por la clase que represente el trámite y debe poder responder al mensaje "double comisión()"

Agregar el una clase para que represente el trámite, que implmeente la interface Comisionable, donde se pueda indicar la descripción del mismo, y el domicilio donde se debe realizar el trámite, la hora en que debe realizarse y la fecha en que el mismo fue realizado.

Modifica en la clase Cadete:

- La lista de tareas a realizar ahora no solo tendrá pedidos, sino también trámites (usar la interface comisionable).
- Modificar el método que calcula la comisión a cobrar para incluir los trámites.