Programación II

Ejercicio Evaluable 2: NowMeal Lite

Dpto. LSIIS. Unidad de Programación

Objetivo: El objetivo de este ejercicio entregable es la familiarización del alumno con el manejo de listas de objetos (ArrayList) y la utilización de JUnitTests.

Entrega: El ejercicio se entregará a través del servidor web de la asignatura:

http://triqui2.fi.upm.es/entrega/

El ejercicio entregado debe compilar en la versión 1.8 del J2SE de Oracle/Sun. Por el hecho de que el ejercicio sea admitido, eso no implicará que el ejercicio esté aprobado.

Grupos: El ejercicio se podrá realizar en grupos de dos alumnos o de forma individual. Se proporcionará al alumno el código que se necesita importar para que tras la entrega se pueda validar la corrección de los datos del autor o autores. Al comienzo del código realizado se deberán completar los datos del alumno o alumnos según este formato:

```
import anotacion.*;
@Programacion2 (
        nombreAutor1 = "nombre",            // (del alumno 1)
        apellidoAutor1 = "apellido1 apellido2",  // (del alumno 1)
        emailUPMAutor1 = "usr@alumnos.upm.es",   // (del alumno 1)
        nombreAutor2 = "nombre",            // (del alumno 2 si lo hay)
        apellidoAutor2 = "apellido1 apellido2",  // (del alumno 2 si lo hay)
        emailUPMAutor2 = "usr@alumnos.upm.es"    // (del alumno 2 si lo hay)
)
public class Moto {

        // .... rellenar aquí el código solicitado en el ejercicio

} // de clase Moto
```

Evaluación: En el momento de entregar el ejercicio en la web de la asignatura, será sometido a una serie de pruebas. Para que la entrega sea admitida, tendrá que superar las pruebas obligatorias. Dependiendo de las pruebas que consiga superar el ejercicio, se obtendrá una nota, que se mostrará al alumno tras realizar la entrega.

Problema a Resolver

NowMeal es una aplicación para la venta de comida a domicilio. Gracias a esta aplicación, los clientes pueden pedir comida a domicilio por medio de un dispositivo móvil. Esta aplicación ofrece a los clientes una lista de restaurantes, los cuales, a su vez, ofrecen distintos tipos de comidas. Al pedir la comida, el cliente debe indicar el restaurante y los platos de comida que desea recibir de ese restaurante.

Cada vez que llega un pedido, NowMeal le debe asignar una moto. Esta moto debe ir primero a recoger la comida al restaurante y luego debe llevar esta comida al cliente. De entre las motos que trabajan para NowMeal y que no están ya asignadas a un pedido, NowMeal debe elegir aquella moto que minimice el coste del pedido para la empresa.

El coste de un pedido se calcula sumando el importe de los platos de comida a los costes del transporte. A su vez, los costes del transporte se calculan sumando el coste que supone para la moto ir desde su posición actual hasta el restaurante y el coste que supone para la moto ir desde el restaurante hasta el domicilio del cliente.

Una vez que la moto ha entregado el pedido al cliente, se le notifica a la aplicación NowMeal. En ese momento, esta moto pasa a estar disponible de nuevo.

Los objetos moto, cliente y restaurante poseen un atributo *codigo* que los identifica. No puede haber dos objetos con el mismo código.

Las posiciones de las motos, clientes y restaurantes se encuentran representadas dentro de la clase Mapa. Gracias a esto, la clase Mapa puede proporcionar un método *distancia(codSitioOrg, codSitioDest)* que devuelve la distancia que debe cubrir una moto para llegar desde un sitio a otro del mapa. Al llamar a este método, cada sitio del mapa se especifica mediante el código del objeto (moto, cliente o restaurante) que lo ocupa.

Al final del enunciado se puede encontrar el diagrama de clases de este ejercicio. El alumno solo tendrá que implementar algunos de los métodos de las clases con fondo amarillo.

Se pide:

1. Implementa el método *coste(origen, destino)* de la clase Moto. El coste es igual a *distancia(origen, destino)*eurosPKm*, siendo *eurosPKm* el precio por kilómetro recorrido que ha negociado esa moto con la empresa. Implementa también el método *coste (destino)* en la clase Moto de forma que el origen sea el código de this. **(2 puntos)**

2. Implementa el constructor de la clase Pedido de manera que inicialice todos y cada uno de los atributos de la clase. **(2 puntos)**

3. Implementa el método *coste(moto)* de la clase Pedido de manera que calcule dicho coste como se ha explicado anteriormente. **(1,5 puntos)**

4. Implementa el método *asignarPedido(pedido)* de la clase GestionRepartoLocal. Este método debe asignar al pedido aquella moto que minimice el coste del pedido para la empresa. A partir del momento en el que se asigne una moto a un pedido, esta misma

moto dejará de estar disponible para otro pedido. Si no existe ninguna moto disponible, este método no hace nada. **(3 puntos)**

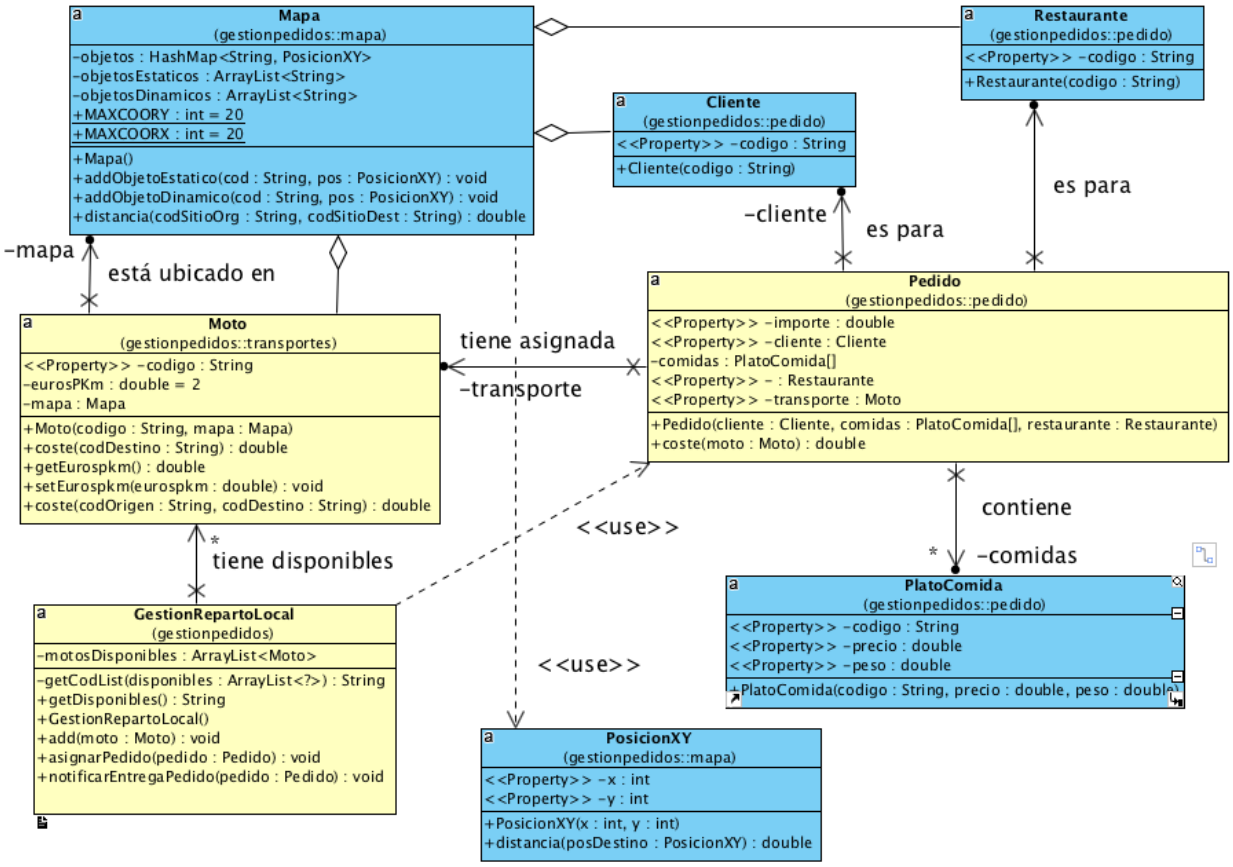
- Para asignar una moto al pedido se debe utilizar el método *setTransporte(moto)* de la clase Pedido.

5. Implementa el método *notificarEntregaPedido(pedido)* de la clase GestionRepartoLocal de manera que añada la moto asignada al pedido a la lista de motos disponibles. **(1,5 puntos)**

Para probar los métodos añadidos a las clases Moto, Pedido y GestionRepartoLocal se proporcionan los JUnitTests MotoJUnitTest, PedidoJUnitTest y GestionRepartoLocalJUnitTest, respectivamente.

Para que se admita la entrega, el alumno tendrá que haber implementado correctamente los métodos de los apartados 1 y 2 (pruebas obligatorias).

Diagrama de clases



NOTA: los atributos marcados como <<Property>> son atributos para los cuales existen getters y setters. En aras de la claridad, estos getters y setters se han omitido.