Ejercicio 2.5 Envío de pedidos


```
public class Supermercado {
    public void notificarPedido(long nroPedido, Cliente cliente) {
        String notificacion = MessageFormat.format("Estimado cliente, se le info
            hemos recibido su pedido con número {0}, el cual será enviado a la d
            {1}", new Object[] { nroPedido, cliente.getDireccionFormateada() });

        // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
        System.out.println(notificacion);
    }
}

public class Cliente {
  public String getDireccionFormateada() {
        return this.direccion.getLocalidad() + ", " +
            this.direccion.getCalle() + ", " +
            this.direccion.getNumero() + ", " +
            this.direccion.getDepartamento();
}
```

Paso uno: Análisis

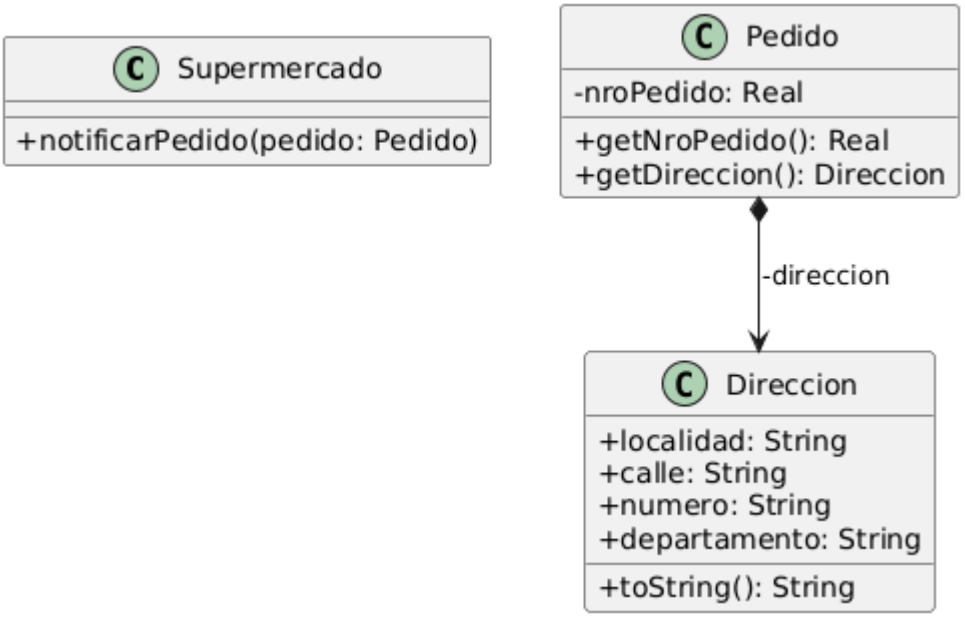
- El método **notificarPedido()** utiliza un vector de objetos para imprimir en consola el número de pedido y la dirección formateada del cliente, cuando podría mas fácil crear una clase *Pedido* que contenga el número del pedido junto a la dirección del cliente. De esta forma, también nos ahorramos la necesidad de tener que pasar el cliente por parámetro.
- La clase **Cliente** tiene un método que se encarga de generar una cadena de texto para la dirección. Esto lo podría y debería hacer la clase **Dirección** misma, y de paso estaríamos aprovechando el polimorfismo (redefinir el método **toString()** de todos los objetos de Java)

Paso dos: Refactoring

1. **Code Smell:** Obsesión por los Primitivos → Se usan primitivos para modelar algo que necesita una estructura mas adecuada
   a. **Refactoring a aplicar:** Replace Data Value with Object → Crear una nueva clase *Pedido* que tenga una variable *nroPedido* y una variable *direccion*. Cambiar la referencia en el método **notificarPedido()**
2. **Code Smell:** Envidia de Atributos → La clase **Cliente** se encarga de formatear el contenido de la clase **Direccion**
   a. **Refactoring a aplicar:** Move Function → Mover el método **getDireccionFormateada()** a la clase **Direccion**.
   b. **Refactoring a aplicar:** Change Function Declaration → Cambiar el nombre del método a *toString()* para aprovechar las funcionalidades de Java.
3. **Code Smell:** Elemento Ocioso → La clase **Cliente** no aporta ninguna utilidad ahora
   a. **Refactoring a aplicar:** Remove Middle Man → Borrar clase Cliente

      // No estoy seguro de este refactoring porque no sé si quieren que borremos la clase. Lo mismo con el scope de las variables. Consultar

Resultado final




```
public class Supermercado {
      public void notificarPedido(Pedido pedido) {
        String notificacion = MessageFormat.format("Estimado cliente, se le info
              + "hemos recibido su pedido con número {0}, " +
            + "el cual será enviado a la dirección {1}",
            pedido.getNroPedido(), pedido.getDireccion());

        // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
       System.out.println(notificacion);
     }
}

public class Pedido {
   private long nroPedido;
   private Direccion direccion;

   public long getNroPedido() {
      return nroPedido;
}

public class Direccion {
   public String localidad;
   public String calle;
```

```
    public String numero;
    public String departamento;

    @Override
    public String toString() {
        return this.localidad + ", "
                + this.calle + ", "
                + this.numero + ", "
                + this.departamento;
    }
}
```