Conclusiones del trabajo práctico de Persistencia

Para realizar este trabajo, utilicé el programa Intellij Idea y SpringBoot con las respectivas dependencias vistas en clase.

A partir del diagrama de clases presentado en el trabajo práctico, cree las clases de Cliente, Domicilio, Pedido, Factura, DetallePedido, Producto y Rubro.

Cliente tiene una relación OneToMany unidireccional con Domicilio y Pedido. Pedido tiene una relación OneToOne con Factura y una relación OneToMany con DetallePedido.

Detalle Pedido tiene una relación ManyToOne con Producto y Rubro tiene una relación OneToMany con Producto.

Luego en el main cree objetos con el método build a modo ejemplo para poder persistirlos en la base de datos H2:

```
Domicilio domicilio1 = Domicilio.builder()
      .calle("Calle1")
      .numero("30")
      .localidad("Gllen")
      .build();

Domicilio domicilio2 = Domicilio.builder()
      .calle("Calle2")
      .numero("30")
      .localidad("GC")
      .build();

Cliente cliente = Cliente.builder()
      .nombre("Juan")
      .apellido("Pérez")
      .telefono("111")
      .email("juan@gmail.com")
      .build();

SimpleDateFormat  formatoFecha = new
SimpleDateFormat("yyyy-MM-dd");
String fecha1String = "2022-04-02";
String fecha2String = "2022-11-30";
String fecha3String = "2022-12-01";
String fecha4String = "2022-12-03";
Date fecha1 = formatoFecha.parse(fecha1String);
Date fecha2 = formatoFecha.parse(fecha2String);
Date fecha3 = formatoFecha.parse(fecha3String);
Date fecha4 = formatoFecha.parse(fecha4String);
```

```
Pedido pedido1 = Pedido.builder()
      .estado(Pedido.Estado.iniciado)
      .fecha(fecha1)
      .tipoEnvio(Pedido.TipoEnvio.retira)
      .total(23)
      .build();

Pedido pedido2 = Pedido.builder()
      .estado(Pedido.Estado.entregado)
      .fecha(fecha2)
      .tipoEnvio(Pedido.TipoEnvio.delivery)
      .total(29)
      .build();



Factura factura1 = Factura.builder()
      .numero(01)
      .fecha(fecha3)
      .descuento(90)
      .formaPago(Factura.FormaPago.efectivo)
      .total(800)
      .build();

Factura factura2 = Factura.builder()
      .numero(02)
      .fecha(fecha4)
      .descuento(50)
      .formaPago(Factura.FormaPago.etc)
      .total(500)
      .build();

DetallePedido detallePedido1 = DetallePedido.builder()
      .cantidad(400)
      .subtotal(2000)
      .build();

DetallePedido detallePedido2 = DetallePedido.builder()
      .cantidad(200)
      .subtotal(3000)
      .build();

Producto producto1 = Producto.builder()
      .tipo(Producto.Tipo.manufacturado)
      .tiempoEstimadoCocina(20)
      .denominacion("A")
      .precioVenta(200)
      .precioCompra(100)
```

```
        .stockActual(40)
        .stockMinimo(10)
        .unidadMedida("cm")
        .receta("a")
        .build();

Producto producto2 = Producto.builder()
        .tipo(Producto.Tipo.insumo)
        .tiempoEstimadoCocina(25)
        .denominacion("B")
        .precioVenta(240)
        .precioCompra(150)
        .stockActual(50)
        .stockMinimo(20)
        .unidadMedida("cm")
        .receta("b")
        .build();

Rubro rubro1 = Rubro.builder()
        .denominacion("casa")
        .build();

Rubro rubro2 = Rubro.builder()
        .denominacion("otros")
        .build();
```

Luego, para lograr la unidireccionalidad, utilicé los metodos set y métodos que cree para poder generar las relaciones entre los objetos:

```
cliente.agregarDomicilio(domicilio1);
cliente.agregarDomicilio(domicilio2);
cliente.agregarPedidos(pedido1);
cliente.agregarPedidos(pedido2);
pedido1.setFactura(factura1);
pedido1.agregarDetallesPedido(detallePedido1);
pedido1.agregarDetallesPedido(detallePedido2);
```

```
rubro1.agregarProductos(producto1);
rubro1.agregarProductos(producto2);
```

El siguiente paso fue guardar los objetos en los repositorios que cree con interfaces java, utilizando el método save:

```
clienteRepository.save(cliente);
detallePedidoRepository.save(detallePedido1);
detallePedidoRepository.save(detallePedido2);
domicilioRepository.save(domicilio1);
domicilioRepository.save(domicilio2);
```

```
facturaRepository.save(factura1);
facturaRepository.save(factura2);
pedidoRepository.save(pedido1);
pedidoRepository.save(pedido2);
rubroRepository.save(rubro1);
rubroRepository.save(rubro2);
```

```
rubro1.agregarProductos(producto1);
rubro1.agregarProductos(producto2);
```

Por último, decidí probar algunos métodos de las consultas JPA:

Mostrar los productos del rubro1:

```
Rubro rubroRecuperado =
rubroRepository.findById(rubro1.getId()).orElse(null);

if (rubroRecuperado != null) {
   rubroRecuperado.mostrarProductos();
}
```

Productos del rubro: casa:
Nombre: A
Nombre: B

Mostrar los pedidos y domicilios del cliente 1:

```
Cliente clienteRecuperado =
clienteRepository.findById(cliente.getId()).orElse(null);

if (clienteRecuperado != null) {
   clienteRecuperado.mostrarPedidos();
   clienteRecuperado.mostrarDomicilios();
}
```

Pedidos de JuanPérez:
ID: 1Estado: iniciado, Fecha: 2022-04-02, Tipo de envío: retira, Total: 23.0
ID: 2Estado: entregado, Fecha: 2022-11-30, Tipo de envío: delivery, Total: 29.0
Domicilios de JuanPérez:
Calle: Calle1, Número: 30, Localidad: GIlen
Calle: Calle2, Número: 30, Localidad: GC

Cambiar el valor de un atributo de producto2 y mostrarlo en pantalla:

```
producto2.setReceta("c");
productoRepository.save(producto2);
```

```
Producto productoRecuperado =
productoRepository.findById(producto2.getId()).orElse(null);
if (productoRecuperado != null) {
   System.out.println("receta " + productoRecuperado.getReceta());
}
```

receta c

Borrar un domicilio de cliente:

```
domicilioRepository.delete(domicilio2);


Cliente clienteRecuperado2 =
clienteRepository.findById(cliente.getId()).orElse(null);

if (clienteRecuperado2 != null) {
   clienteRecuperado2.mostrarDomicilios();
}
```

Domicilios de JuanPérez:
Calle: Calle1, Número: 30, Localidad: GIlen

Es de destacar que tuve dos problemas a la hora de mapear este modelo:

El primero fue que para crear una relación ManyToOne entre DetallePedido y Producto, previamente se debe persistir el producto creado en la base de datos para luego poder guardarlo en DetallePedido con el método set. Quedando de la siguiente forma:

```
productoRepository.save(producto1);
productoRepository.save(producto2);

detallePedido1.setProducto(producto1);
```

Por otro lado, tuve un problema en la relación ManyToOne de rubro con producto, ya que no utilicé `fetch = FetchType.EAGER,` por lo que me generaba un error de carga perezosa.

Para finalizar el informe, es de mencionar que se puede levantar la base de datos H2 con http://localhost:8080/h2-console/ y verificar que los datos ingresados están correctamente mapeados.