Refactoring 4

- Estudiante: Javier López
- Problema: Problema 3

Código Inicial

```java
// Inventory.java
package refactoring.problema3;

import refactoring.problema3.Product;
import refactoring.problema3.Sale;
import refactoring.problema3.Order;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class Inventory {

    public static void main(String[] args) {
        String csvFileProducts = "./refactoring/problema3/da
        String csvFileSales = "./refactoring/problema3/data/
        String csvFileOrders = "./refactoring/problema3/dat

        System.out.println(csvFileProducts);
        String csvSplitBy = ",";

        ArrayList<Product> products = new ArrayList<Product>
        ArrayList<Sale> sales = new ArrayList<Sale>();
        ArrayList<Order> orders = new ArrayList<Order>();

        try (BufferedReader br = new BufferedReader(new File
```

```
        String line = br.readLine();

        while ((line = br.readLine()) != null) {
            String[] data = line.split(csvSplitBy);

            // Access the product data
            int itemId = Integer.parseInt(data[0]);
            String item = data[1];
            int quantity = Integer.parseInt(data[2]);

            products.add(new Product(itemId, item, quant
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    try (BufferedReader br = new BufferedReader(new File
        String line = br.readLine();

        while ((line = br.readLine()) != null) {
            String[] data = line.split(",");

            int saleId = Integer.parseInt(data[0].trim(
            String saleDate = data[1].trim();
            int itemId = Integer.parseInt(data[2].trim(
            int quantity = Integer.parseInt(data[3].trir

            Sale sale = new Sale(saleId, saleDate, item
            sales.add(sale);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    try (BufferedReader br = new BufferedReader(new File
        String line = br.readLine();

        while ((line = br.readLine()) != null) {
            String[] data = line.split(",");

            int orderId = Integer.parseInt(data[0].trim(
```

```
                String orderDate = data[1].trim();
                int itemId = Integer.parseInt(data[2].trim(
                int quantity = Integer.parseInt(data[3].trir

                Order order = new Order(orderId, orderDate,
                orders.add(order);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (Order order : orders) {
            Product item = products.get(order.getItemId());
            item.setQuantity(item.getQuantity() + order.get(
        }

        for (Sale sale : sales) {
            Product item = products.get(sale.getItemId());
            item.setQuantity(item.getQuantity() - sale.getQu
        }

        for (Product product : products) {
            System.out.println(product.getItem() + " " + pr(
        }

    }
}

// Product.java
package refactoring.problema3;

public class Product {
    private int itemId;
    private String item;
    private int quantity;

    public Product(int itemId, String item, int quantity) {
        this.itemId = itemId;
        this.item = item;
        this.quantity = quantity;
    }
```

```java
    public int getItemId() {
        return itemId;
    }

    public void setItemId(int itemId) {
        this.itemId = itemId;
    }

    public String getItem() {
        return item;
    }

    public void setItem(String item) {
        this.item = item;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}

// Order.java
package refactoring.problema3;

public class Order {
    private int orderId;
    private String orderDate;
    private int itemId;
    private int quantity;

    public Order(int orderId, String orderDate, int itemId,
        this.orderId = orderId;
        this.orderDate = orderDate;
        this.itemId = itemId;
        this.quantity = quantity;
    }
```

```
    public int getOrderId() {
        return orderId;
    }

    public String getOrderDate() {
        return orderDate;
    }

    public int getItemId() {
        return itemId;
    }

    public int getQuantity() {
        return quantity;
    }
}


// Sale.java
package refactoring.problema3;

public class Sale {
    private int saleId;
    private String saleDate;
    private int itemId;
    private int quantity;

    public Sale(int saleId, String saleDate, int itemId, int
        this.saleId = saleId;
        this.saleDate = saleDate;
        this.itemId = itemId;
        this.quantity = quantity;
    }

    public int getSaleId() {
        return saleId;
    }

    public String getSaleDate() {
        return saleDate;
```

```
    }

    public int getItemId() {
        return itemId;
    }

    public int getQuantity() {
        return quantity;
    }
}
```

Solución

```
// Inventory.java
package refactoring.problema3_new;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Inventory {

    @FunctionalInterface
    interface DataConstructor<T> {
        T create(String[] data);
    }

    private static final String CSV_FILE_PRODUCTS = "./refa
    private static final String CSV_FILE_SALES = "./refacto
    private static final String CSV_FILE_ORDERS = "./refact
    private static final String CSV_SPLIT_BY = ",";

    public static void main(String[] args) {
        List<Product> products = loadProducts(CSV_FILE_PROD
```

```
        List<Sale> sales = loadSales(CSV_FILE_SALES);
        List<Order> orders = loadOrders(CSV_FILE_ORDERS);

        updateInventory(products, orders, sales);
        printInventory(products);
    }

    private static <T> List<T> loadData(String filePath, Da
        try {
            return Files.lines(Paths.get(filePath))
                        .skip(1)
                        .map(line -> line.split(CSV_SPLIT_B
                        .map(constructor::create)
                        .collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
            return new ArrayList<>();
        }
    }

    private static List<Product> loadProducts(String filePa
        return loadData(filePath, Product::fromString);
    }

    private static List<Sale> loadSales(String filePath) {
        return loadData(filePath, Sale::fromString);
    }

    private static List<Order> loadOrders(String filePath)
        return loadData(filePath, Order::fromString);
    }

    private static void updateInventory(List<Product> produ
        Map<Integer, Product> productMap = products.stream(
                .collect(Collectors.toMap(Product::getItemI

        for (Order order : orders) {
            Product item = productMap.get(order.getItemId()
            if (item != null) {
                item.setQuantity(item.getQuantity() + order
            } else {
```

```
                throw new RuntimeException("Product not fou
            }
        }

        for (Sale sale : sales) {
            Product item = productMap.get(sale.getItemId())
            if (item != null) {
                item.setQuantity(item.getQuantity() - sale.g
            } else {
                throw new RuntimeException("Product not fou
            }
        }
    }

    private static void printInventory(List<Product> product
        products.forEach(product -> System.out.println(produ
    }

}

// Product.java
package refactoring.problema3_new;

public class Product {
    private int itemId;
    private String item;
    private int quantity;

    public Product(int itemId, String item, int quantity) {
        this.itemId = itemId;
        this.item = item;
        this.quantity = quantity;
    }

    public static Product fromString(String[] productString)
      return new Product(
        Integer.parseInt(productString[0]),
        productString[1],
        Integer.parseInt(productString[2])
      );
    }
```

```java
    public int getItemId() {
        return itemId;
    }

    public void setItemId(int itemId) {
        this.itemId = itemId;
    }

    public String getItem() {
        return item;
    }

    public void setItem(String item) {
        this.item = item;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}

// Order.java
package refactoring.problema3_new;

public class Order {
    private int orderId;
    private String orderDate;
    private int itemId;
    private int quantity;

    public Order(int orderId, String orderDate, int itemId,
        this.orderId = orderId;
        this.orderDate = orderDate;
        this.itemId = itemId;
        this.quantity = quantity;
    }
```

```java
    public static Order fromString(String[] orderString) {
      return new Order(
        Integer.parseInt(orderString[0]),
        orderString[1],
        Integer.parseInt(orderString[2]),
        Integer.parseInt(orderString[3])
      );
    }

    public int getOrderId() {
        return orderId;
    }

    public String getOrderDate() {
        return orderDate;
    }

    public int getItemId() {
        return itemId;
    }

    public int getQuantity() {
        return quantity;
    }
}

// Sale.java
package refactoring.problema3_new;

public class Sale {
    private int saleId;
    private String saleDate;
    private int itemId;
    private int quantity;

    public Sale(int saleId, String saleDate, int itemId, int
        this.saleId = saleId;
        this.saleDate = saleDate;
        this.itemId = itemId;
        this.quantity = quantity;
```

```
    }

    public static Sale fromString(String[] saleString) {
      return new Sale(
        Integer.parseInt(saleString[0]),
        saleString[1],
        Integer.parseInt(saleString[2]),
        Integer.parseInt(saleString[3])
      );
    }

    public int getSaleId() {
        return saleId;
    }

    public String getSaleDate() {
        return saleDate;
    }

    public int getItemId() {
        return itemId;
    }

    public int getQuantity() {
        return quantity;
    }
}
```

Refactorings Utilizados

Extract Function

Se descompuso el método main de producto en múltiples funciones (métodos).

- loadData: Se encarga de cargar los datos de un archivo csv dado una función (DataConstructor).

- loadProducts: Se encarga de cargar los productos de un archivo csv.
- loadSales: Se encarga de cargar las ventas de un archivo csv.
- loadOrders: Se encarga de cargar las ordenes de un archivo csv.
- updateInventory: Se encarga de hacer el proceso de actualización de inventario.
- printInventory: Se encarga de imprimir el inventario.

Esta división de funcionalidades claramente hace más fácil de modificar y mantener el código, es también mucho más fácil de entender.

Aparte de esto se implemento una factory function que hace la tarea de instanciar correctamente las clases de producto, venta y orden. Esto es también una función que fue extraída.

Replace Constructor with Factory Function

Las clases de producto, venta y orden fueron modificadas para que tengan un método estático que se encargue de crear una instancia de la clase a partir de un arreglo de strings leídos del csv, esto permite que la misma clase haga la construción de su instancia y de esa forma no tiene que exponer esos detalles a partes del código externas.