Finvory System Walkthrough Report
The POOwer Rangers of Programming

FINVORY

Objective

The purpose of this walkthrough is to review the design and functionality of the Finvory System to identify possible improvements and ensure that it meets the project features.

CASE 1

- **Roles**
  Author: Arelys Otavalo
  Reviewer: Mathews Pastor
  Scribe: Maryuri Quiña
  Moderator: Joseph Medina

- **Description of the Artifact**
  The review focused on verifying compliance with the principles of Object-Oriented Programming (OOP), specifically encapsulation in addition to Clean Code standards, as well as identifying defects in user interaction with the messages displayed on the console

  The following components were examined:

- **OOP Classes:**
  The implementation of the CompanyAccount class (including attributes, constructors, and access methods), along with code snippets from helper methods such as findProduct and findCustomer. The review specifically focused on the correct implementation of getters and setters, as well as the use of Clean Code principles.

- **User Interface Flows:**
  Screenshots and code snippets illustrating key user interactions, including:

    - The *Personal Account Registration* process (data display, success message, and character encoding).
    - The *Product Deletion* flow (missing confirmation prompt).
    - The *Client Search and Registration* process during a new sale (redundant behavior).

- **Utility Methods:**
  A code snippet of the askQuantity method used for input validation in sales and purchase transactions.

- **Comments and Findings**
  - Missing implementation of getters and setters for the basic class attributes (name, address, ruc, phone, email). This violates POO encapsulation principles. All missing getter and setter methods for the private attributes of the CompanyAccount class must be implemented.

```java
package ec.espe.edu.finvory.model;
public class CompanyAccount {
    private String name;
    private Address address;
    private String ruc;
    private String phone;
    private String email;

    public CompanyAccount() {}

    public CompanyAccount(String name, Address address, String ruc, String phone, String email) {
        this.name = name;
        this.address = address;
        this.ruc = ruc;
        this.phone = phone;
        this.email = email;
    }

}
```

  - Failure to comply with Clean Code standards in the structure and ordering of getters and setters. Methods are scattered, reducing code readability and maintenance. Refactor getters and setters to follow the standard conventions and maintain a consistent order corresponding to attribute declaration.

```java
public ArrayList<Customer> getCustomers() { return customers; }
public ArrayList<Product> getProducts() { return products; }
public ArrayList<Supplier> getSuppliers() { return suppliers; }
public ArrayList<Inventory> getInventories() { return inventories; }
public ArrayList<PersonalAccount> getPersonalAccounts() { return personalAccounts; }
public InventoryOfObsolete getObsoleteInventory() { return obsoleteInventory; }
public ArrayList<InvoiceSim> getInvoices() { return invoices; }
public ArrayList<ReturnedProduct> getReturns() { return returns; }
public float getTaxRate() { return taxRate; }
public void setTaxRate(float taxRate) { this.taxRate = taxRate; }
public AdminDetail getAdminInfo() { return adminInfo; }
public void setAdminInfo(AdminDetail adminInfo) { this.adminInfo = adminInfo; }
public void setCompanyInfo(CompanyAccount companyInfo) { this.companyInfo = companyInfo; }
public CompanyAccount getCompanyInfo() { return companyInfo; }
public float getProfitPercentage() { return profitPercentage; }
public void setProfitPercentage(float profitPercentage) { this.profitPercentage = profitPercentage; }
public float getDiscountStandard() { return discountStandard; }
public void setDiscountStandard(float discountStandard) { this.discountStandard = discountStandard; }
public float getDiscountPremium() { return discountPremium; }
public void setDiscountPremium(float discountPremium) { this.discountPremium = discountPremium; }
public float getDiscountVip() { return discountVip; }
public void setDiscountVip(float discountVip) { this.discountVip = discountVip; }
```

  - Implement Clean Code in the getters and setters according to the order in which the code should be generated.

- Character Encoding Error in Console. An unreadable and unknown character appears before the message "Personal Account successfully registered!", indicating a character encoding problem. The character encoding of the console or source code must be reviewed and standardized.

```
--- REGISTRO DE CUENTA PERSONAL ---
-> Nombre Completo: Arelys
-> Nombre de Usuario (para login): Are
-> Contrasena: 1234
INFO: ¦Cuenta Personal registrada con exito!
INFO: Ahora puede iniciar sesion.

--- TIPO DE REGISTRO ---
1. Registrar Cuenta de Compania (Admin)
2. Registrar Cuenta Personal (Invitado)
0. Volver
Seleccione una opcion:
```

- Lack of confirmation for a destructive action (Eliminar Producto). The system immediately requests the product ID (KTY) for deletion without a preceding confirmation prompt

```
1. Ver Lista de Productos
2. Editar Producto
3. Eliminar Producto
4. Registrar Devolucion (a Obsoletos)
5. Gestionar Stock Obsoleto
0. Volver al Menu Principal
Seleccione una opcion: 3
-> Ingrese ID del producto (o 'fin' para terminar): KTY
```

- Redundancy in the confirmation prompt. The confirmation option to register a new client is displayed repeatedly as (s/n) (s/n), which is unnecessary and confusing for the user. The repeated text should be removed, leaving the confirmation prompt with only one instance.

```
--- MENU PRINCIPAL (Company Account) ---
1. Nueva Venta (Facturar)
2. Gestionar Inventario y Productos
3. Gestionar Clientes
4. Gestionar Proveedores
5. Administracion
6. Ver Reportes y Dashboard
7. Generar Cotizacion
0. Guardar y Salir
Seleccione una opcion: 1
-> Busque al cliente (Nombre o ID): mARYURI
ERROR: Cliente 'mARYURI' no encontrado.
¦Desea registrarlo como nuevo cliente ahora? (s/n) (s/n): S
```

CASE 2

- **Roles**
  Author: Mathews Pastor
  Reviewer: Arelys Otavalo
  Scribe: Joseph Medina
  Moderator: Maryuri Quiña

- **Description of the Artifact**
  In this method, the concepts of clean code were specifically put into practice; in this situation, this involved using correct descriptive variable names that were relevant to the method's action, as well as removing unnecessary comments.

```
public int askQuantity(int maxStock) {
    int qty = -1;
    while (qty < 0 || qty > maxStock) {
        qty = getPositiveIntInput("-> Cantidad (Max: " + maxStock + "): ");
        if (qty > maxStock) {
            showError("Stock insuficiente.");
            qty = -1;
        }
    }
    return qty;
}
```

- The problem with abbreviations (like qty, inv, dStd) is that they force the programmer reading your code to do a mental translation. Every time they see dStd, they have to stop and think: "Ah, d is for discount and Std is for Standard."

```
private Customer findCustomer(String query) {
    if (query == null || query.isEmpty()) return null;
    query = query.toLowerCase();
    for (Customer c : data.getCustomers()) {
        if (query.equals(c.getIdentification()) ||
            (c.getName() != null && c.getName().toLowerCase().contains(query))) {
            return c;
        }
    }
    return null;
}
```

- Violation of the "Descriptive Names" Principle: The code uses Customer c as the variable name in the for loop. This is a single-letter name that does not reveal the intent. Following the principles of "Clean Code," it should be renamed to customer or customerToCheck to make the code self-explanatory, eliminating the need for the reader to "mentalize" the code.

- **Comments and Findings**

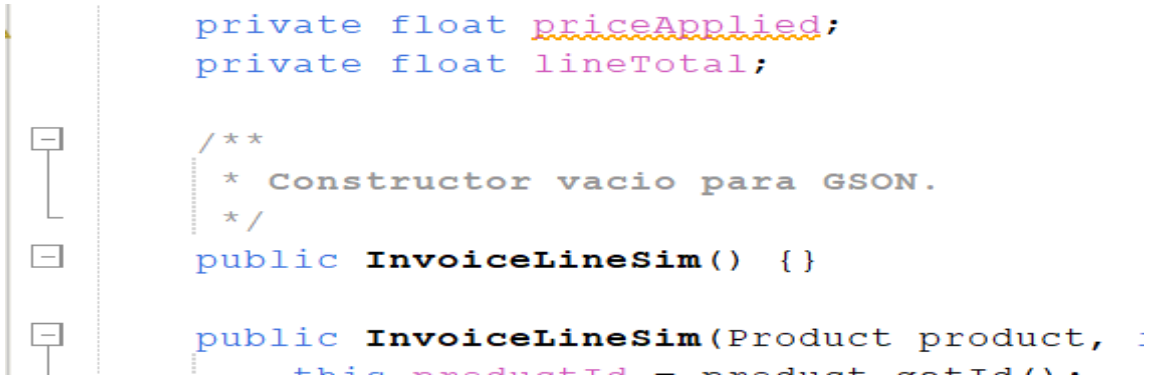
- Clean Code Is Not Optional, It's Essential
- During the walkthrough, we identified several "code smells" that violated Clean Code principles.
- Avoid Abbreviations: We discovered that using single-letter variable names (like Product p or Customer c) or abbreviations (like qty or dStd) saves time when writing, but costs a lot of time when reading.
- Descriptive Names: By changing variables to descriptive names (like productToEdit, quantity, discountStandard), the code becomes self-documenting. This made complex logic (like in handleNewSale or handleCreateProduct) instantly readable.

CASE 3

- **Roles**
  Author: Mathews Pastor
  Reviewer: Joseph Medina
  Scribe: Maryuri Quiña
  Moderator: Arelys Otavalo

- **Description of the Artifact**
  The walkthrough focused on reviewing a specific fragment of the InvoiceLineSim class, particularly the default constructor and its associated comment block. The comment /** Empty constructor for GSON. */ was identified as unnecessary and redundant, since the constructor's purpose is already clear from its implementation. This observation aimed to ensure adherence to Clean Code principles and improve overall code readability.

- **Comments and Findings**
  - Unnecessary and Redundant Comment. The comment /** Empty constructor for GSON. */ on the public InvoiceLineSim() {} method is redundant and adds no value. The unnecessary comment block should be removed.



CASE 4

- **Roles**
  Author: Mathews Pastor
  Reviewer: Maryuri Quiña
  Scribe: Arelys Otavalo
  Moderator: Joseph Medina

- **Description of the Artifact:**
  The loadCustomersCsv method was reviewed, with a specific focus on variable declarations within the try-with-resources block. The analysis centered on compliance with Clean Code naming conventions.

- **Comments and Findings:**
  - The variable name `br` (for `BufferedReader`) is a cryptic abbreviation. Although its type is inferred from the initialization (`new BufferedReader(...)`), the name alone doesn't reveal the intention and forces the reader to make that mental connection. This reduces code readability. According to Clean Code principles, explicit and descriptive names, such as `bufferedReader` or simply `reader`, should be favored to make the code self-documenting and easier to understand at a glance.

```
private ArrayList<Customer> loadCustomersCsv() {
    ArrayList<Customer> customers = new ArrayList<>();
    File file = new File(CLIENTS_FILE);
    if (!file.exists()) return customers;

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        br.readLine();
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split(DELIMITER);
            if (parts.length == 5) {
                customers.add(new Customer(parts[1], parts[0], parts[2], parts[3], parts[4]));
            }
        }
    } catch (IOException e) {
        System.err.println("Error al cargar " + CLIENTS_FILE + ": " + e.getMessage());
    }
    return customers;
}
```

CASE 5

- **Roles**
  Author: Maryuri Quiña
  Reviewer: Joseph Medina
  Scribe: Mathews Pastor
  Moderator: Arelys Otavalo

- **Description of the Artifact**

The walkthrough focused on examining a code segment related to product transactions, where variable naming conventions were reviewed for compliance with Clean Code principles. The analysis targeted the declaration and usage of the variables p and qty, which serve to reference products and quantities in transaction operations. The review aimed to ensure that the code maintains clarity, consistency, and adherence to best practices for variable naming within the Finvory System.

- **Comments and Findings**
  - A Clean Code issue was found due to the use of cryptic abbreviations in variable names. The variables p (Product) and qty (quantity) reduce readability and make the code less self-explanatory. It is recommended to replace them with clearer names such as product and quantity.

```
public void addLine(Product p, int qty, float priceApplied) {
    lines.add(new InvoiceLineSim(p, qty, priceApplied));
}
```

CASE 6

- **Roles**
  Author: Arelys Otavalo
  Reviewer: Mathews Pastor
  Scribe: Joseph Medina
  Moderator: Maryuri Quiña

- **Description of the Artifact:**
  The askReturnReason utility method was examined by analyzing the user interaction flow. Specifically, the messages printed to the console were reviewed to verify language consistency within the user interface.

- **Comments and Findings:**
  - User Interface Language Inconsistency. The askReturnReason method displays a mixed-language interface to the user. The request title ("Reason for Return") is in Spanish, but the options presented to the user ("DEFECTIVE" and "WRONG_PURCHASE") are in English. This creates a confusing and unprofessional user experience. All user-directed text should be standardized in a single language (in this case, Spanish).

```
public String askReturnReason() {
    System.out.println("-> Motivo de la devolucion:");
    System.out.println("   1. DEFECTIVE (Defectuoso)");
    System.out.println("   2. WRONG_PURCHASE (Compra incorrecta)");
    int opt = getIntInput();
    return (opt == 1) ? "DEFECTIVE" : "WRONG_PURCHASE";
}
```

CASE 7

- **Roles**
  Author: Joseph Medina
  Reviewer: Mathews Pastor
  Scribe: Maryuri Quiña
  Moderator: Arelys Otavalo

- **Description of the Artifact:**
  The FinvoryController class was reviewed, focusing on the declaration of its attributes and constructor. The goal was to verify compliance with Clean Code principles, specifically regarding naming conventions.

- **Comments and Findings:**
  - In the FinvoryController class, `db` is a non-descriptive abbreviation. Although `db` is commonly used for "database," it violates the Clean Code principle of using names that reveal intent, thus reducing code readability. The attribute should be renamed to `database` to explicitly declare its purpose and facilitate maintenance.

```
public class FinvoryController {

    private FinvoryData data;
    private FinvoryView view;
    private Database db;

    public FinvoryController(FinvoryData data, FinvoryView view, Database db) {
        this.data = data;
        this.view = view;
        this.db = db;
    }
```