Refactoring

Código de estudiante: 202210180

Refactoring 1:

Problema 1

El archivo principal del problema cuenta con estos síntomas:

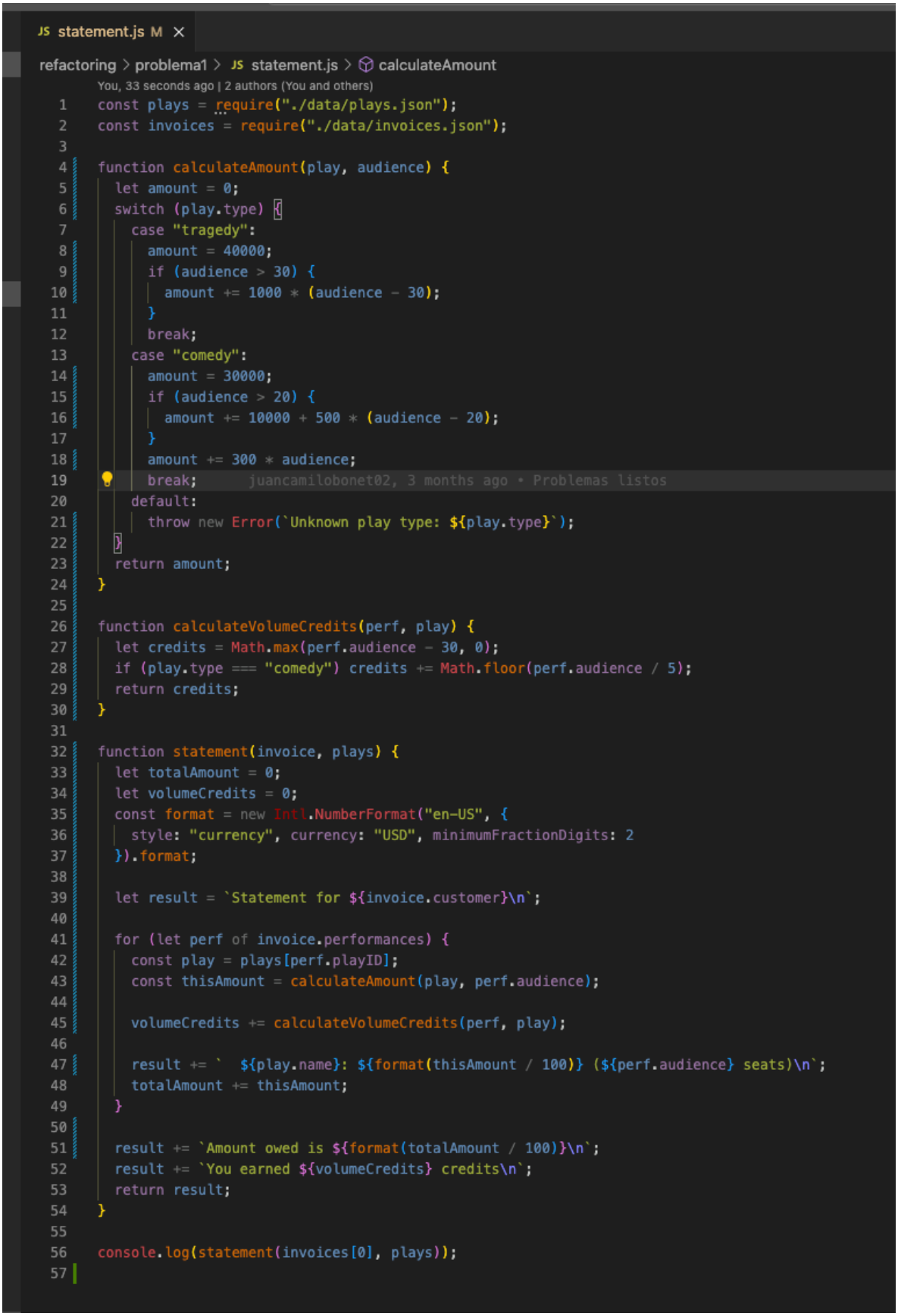
- ☐ Alta complejidad en statement
- ☐ Dispersión de responsabilidades
- ☐ Statement es un método largo con respecto a lo que propone su nombre

```
const plays = require("./data/plays.json");
const invoices = require("./data/invoices.json");

function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
                       { style: "currency", currency: "USD",
                         minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = plays[perf.playID];
    let thisAmount = 0;

    switch (play.type) {
    case "tragedy":
      thisAmount = 40000;
      if (perf.audience > 30) {
```

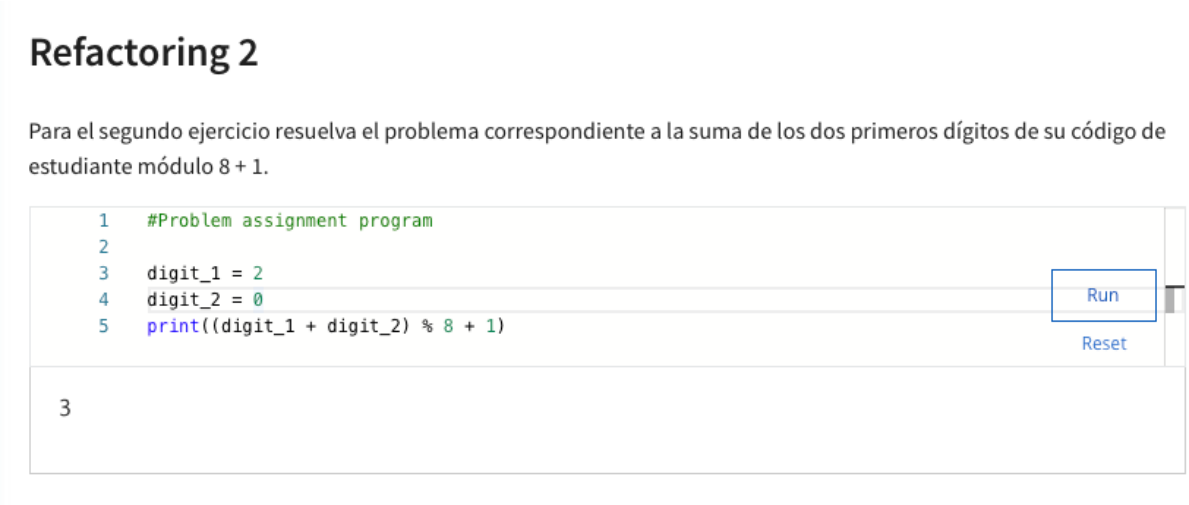
```
        thisAmount += 1000 * (perf.audience - 30);
      }
      break;
    case "comedy":
      thisAmount = 30000;
      if (perf.audience > 20) {
        thisAmount += 10000 + 500 * (perf.audience - 20);
      }
      thisAmount += 300 * perf.audience;
      break;
    default:
        throw new Error(`unknown type: ${play.type}`);
    }

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === play.type) volumeCredits += Math.floor(perf.audience / 5);

    // print line for this order
    result += `  ${play.name}: ${format(thisAmount/100)} (${perf.audience} seats)\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}

console.log(statement(invoices[0], plays));
```

El refactoring sugerido en este caso sería:

1. Extraer en funciones aparte el cálculo de monto por obra y la cantidad de créditos
    a. Mejorando la legibilidad y comprensión del código
    b. Disminuye la complejidad del código al separar su funcionalidad en pequeñas funciones que son más mantenibles.
    c. El manejo de errores se realiza del lado del cálculo de costo, en el caso de que la obra no se conozca y se centraliza en este método.
    d. Statement se encarga de agrupar la información más no de realizar los cálculos independientes o la combinación de datos según la otra.
2. El resultado de dicho refactor, sería el siguiente:


statement.js M
refactoring > problema1 > statement.js > calculateAmount
You, 33 seconds ago | 2 authors (You and others)
const plays = require("./data/plays.json");
const invoices = require("./data/invoices.json");
function calculateAmount(play, audience) {
let amount = 0;
switch (play.type) {
case "tragedy":
amount = 40000;
if (audience > 30) {
amount += 1000 * (audience - 30);
}
break;
case "comedy":
amount = 30000;
if (audience > 20) {
amount += 10000 + 500 * (audience - 20);
}
amount += 300 * audience;
break;
juancamilobonet02, 3 months ago • Problemas listos
default:
throw new Error(`Unknown play type: ${play.type}`);
}
return amount;
}
function calculateVolumeCredits(perf, play) {
let credits = Math.max(perf.audience - 30, 0);
if (play.type === "comedy") credits += Math.floor(perf.audience / 5);
return credits;
}
function statement(invoice, plays) {
let totalAmount = 0;
let volumeCredits = 0;
const format = new Intl.NumberFormat("en-US", {
style: "currency", currency: "USD", minimumFractionDigits: 2
}).format;
let result = `Statement for ${invoice.customer}\n`;
for (let perf of invoice.performances) {
const play = plays[perf.playID];
const thisAmount = calculateAmount(play, perf.audience);
volumeCredits += calculateVolumeCredits(perf, play);
result += `  ${play.name}: ${format(thisAmount / 100)} (${perf.audience} seats)\n`;
totalAmount += thisAmount;
}
result += `Amount owed is ${format(totalAmount / 100)}\n`;
result += `You earned ${volumeCredits} credits\n`;
return result;
}
console.log(statement(invoices[0], plays));

Refactoring 2:



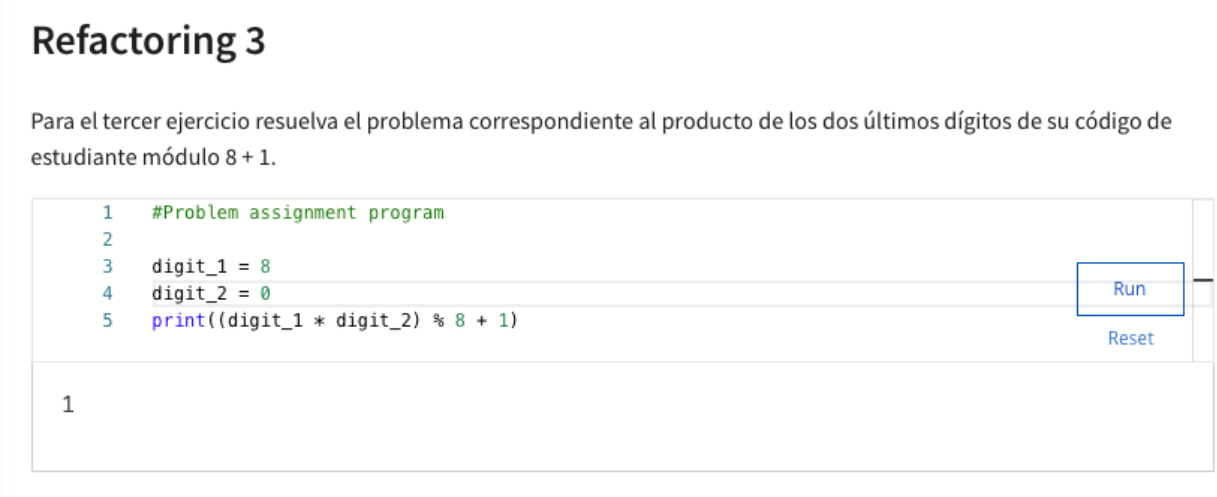
Problema 3

Para este problema el síntoma que más resalta es la duplicación de lógica entre lo que se maneja para Sales y Orders.
El refactoring sugerido sería el siguiente:

- Extraer en funciones el procesamiento de órdenes y ventas.
  - Mejora la legibilidad del código fuente
  - Separa las responsabilidades dentro del método principal.
- Extraer en una función el procesamiento del archivo CSV
  - Centraliza el manejo de los archivos CSV
  - Disminuye la complejidad del método main en la casa inventario
- Obteniendo luego del refactor el código de inventario siguiente:

```java
...
public class Inventory {

    private static final String CSV_SPLIT_BY = ",";

    Run | Debug
    public static void main(String[] args) {
        String csvFileProducts = "./refactoring/problema3/data/products.csv";
        String csvFileSales = "./refactoring/problema3/data/sales.csv";
        String csvFileOrders = "./refactoring/problema3/data/orders.csv";

        Map<Integer, Product> products = readProducts(csvFileProducts);
        List<Sale> sales = readSales(csvFileSales);
        List<Order> orders = readOrders(csvFileOrders);

        updateInventory(products, orders, sales);

        products.values().forEach(product ->
            System.out.println(product.getItem() + " " + product.getQuantity()));
    }

    private static Map<Integer, Product> readProducts(String fileName) {
        Map<Integer, Product> products = new HashMap<>();
        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            br.readLine(); // Skip header
            String line;
            while ((line = br.readLine()) != null) {
                String[] data = line.split(CSV_SPLIT_BY);
                int itemId = Integer.parseInt(data[0].trim());
                products.put(itemId, new Product(itemId, data[1].trim(), Integer.parseInt(data[2].trim())));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return products;
    }

    // Similar methods for readSales and readOrders

    int refactoring.problema3.Order.getItemId()
    getItemId
    private static void updateInventory(Map<In                                    List<Sale> sales) {
        for (Order order : orders) {
            Product item = products.get(order.getItemId());
            item.setQuantity(item.getQuantity() + order.getQuantity());
        }

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

Refactoring 3:



Problema 2

Este problema permite apreciar el siguiente síntoma:

- El código funcional se encuentra en la clase principal de organizar.py

El refactoring sugerido para dar solución a este síntoma sería:

- Realizar la separación de la funcionalidad de lectura de archivos en un método
  - Hace más comprensible el manejo de los archivos y permite realizar cambios sobre el manejo de archivos de forma más sencilla.
  - Separación de responsabilidades
- Realizar el manejo de los errores en los logs de forma separada
  - Permite centralizar el manejo de los errores de los logs en un método aparte.
  - Separación de responsabilidades
- Definición de constantes de los tipos de error que se encontraban en los logs.
  - Permite disminuir los reprocesos al momento de realizar cambios en los tipos de error en los logs.
  - Mejora la legibilidad y comprensión del código fuente.
  - Permite el filtrado según severidad de los registros en el log

- El archivo resultante del refactoring sería el siguiente:

refactoring > problema2 > organizar.py > ...

You, 1 second ago | 2 authors (You and others)

```python
# Tenemos un archivo de logs que contiene errores, warnings y mensajes de información.
# Cada linea del archivo comienza con una letra que indica el tipo de log. Si
# comienza con 'E' es un error, si comienza con 'W' es un warning y si comienza
# con 'I' es de información. Luego de la letra, se encuentra un entero que
# indica el tiempo del log, y luego el mensaje del log. Excepto en el caso de
# los errores, que luego del tipo se encuentra un entero que indica la severidad
# del error. Por ejemplo:

# I 147 iniciando el programa
# W 604 this is not a warning
# E 2 4562 unexpected token

# Queremos separar los errores mas severos (con severidad mayor a 50) y ordenarlos
# cronologicamente. Despues, queremos imprimirlos a la pantalla.

import os
from collections import namedtuple

LogEntry = namedtuple('LogEntry', ['type', 'severity', 'time', 'message'])

def read_log_entries(file_path):
    with open(file_path, 'r') as file:
        for line in file:
            yield line.strip().split(' ')

def filter_severe_errors(log_entries, severity_threshold=50):
    for entry in log_entries:
        if entry[0] == 'E' and int(entry[1]) > severity_threshold:
            yield LogEntry(type=entry[0], severity=int(entry[1]), time=int(entry[2]), message=' '.join(entry[3:]))

def sort_errors_by_time(errors):
    return sorted(errors, key=lambda x: x.time)

def display_errors(errors):
    for error in errors:
        print(f"{error.type} {error.severity} {error.time} {error.message}")

def organize_log_file(error_file_path):
    try:
        log_entries = read_log_entries(error_file_path)
        severe_errors = filter_severe_errors(log_entries)
        sorted_errors = sort_errors_by_time(severe_errors)
        display_errors(sorted_errors)
    except IOError as e:
        print(f"Error reading file: {e}")

def main():
    error_file_path = './refactoring/problema2/data/error.log'
    organize_log_file(error_file_path)

if __name__ == '__main__':
    main()

```

Refactoring 4:

Refactoring 4

Para el cuarto ejercicio resuelva el problema correspondiente a la suma de los dígitos 5, 6, y 7 de su código de estudiante módulo 8 + 1.

```
1  #Problem assignment program
2
3  digit_5 = 1
4  digit_6 = 0
5  digit_7 = 1
6  print((digit_5 + digit_6 + digit_7) % 8 + 1)
```

Run

Reset

```
3
```

Problema 4

Este problema cuenta con los siguientes síntomas:

- ☐ Métodos muy largos
- ☐ Alta complejidad
- ☐ Mezcla de responsabilidades

El refactoring sugerido para solucionar estos síntomas:

- ☐ Extraer los fragmentos que realizan tareas pequeñas en métodos independientes
  - Separar responsabilidades
  - Encapsular funcionalidad
  - Mejor legibilidad
  - Menor complejidad
- ☐ Extraer el manejo de archivos en métodos aparte
  - Encapsulado de responsabilidad
  - Extensibilidad de funcionalidad
- ☐ El resultado del refactoring sería el siguiente:

refactoring > problema4 > todo_list.py > ...

You, 51 seconds ago | 2 authors (You and others)

```python
import os

def read_tasks(file_path):
    try:
        with open(file_path, 'r') as file:
            return [line.strip() for line in file]
    except FileNotFoundError:
        print('No existe el archivo todo.txt')
        return []

def write_tasks(file_path, tasks):
    with open(file_path, 'w') as file:
        for task in tasks:
            file.write(task + '\n')

def display_tasks(tasks):
    print('-------------------------')
    print('Tareas:')
    for i, task in enumerate(tasks):
        print(f'{i + 1}. {task}')

def add_task(tasks):
    task = input('Ingrese la tarea: ')
    tasks.append(task)
    print('Tarea agregada')

def delete_task(tasks):
    task_index = input('Ingrese el número de la tarea a eliminar: ')
    if task_index.isdigit() and (0 < int(task_index) <= len(tasks)):
        tasks.pop(int(task_index) - 1)
        print('Tarea eliminada')
    else:
        print('No existe la tarea')

def display_menu():
    print('-------------------------')
    print('1. Ver tareas')
    print('2. Agregar una tarea')
    print('3. Eliminar una tarea')
    print('4. Salir')
    print('-------------------------')
    return input('Ingrese una opción: ')

def todo_list():
    file_path = "./refactoring/problema4/data/todo.txt"
    tasks = read_tasks(file_path)

    print('Bienvenido a su lista de tareas. Estas son sus opciones:')

    while True:
        option = display_menu()

        if option == '1':
            display_tasks(tasks)
        elif option == '2':
            add_task(tasks)
        elif option == '3':
            delete_task(tasks)
        elif option == '4':
            print('Adiós')
            break

    write_tasks(file_path, tasks)

if __name__ == '__main__':
    todo_list()
```