WORKSHOP #3

Johan Esmit Sichacá González - 20242020313

Sergio Andres Diaz Cuervo - 20251020166

John Mario Jimenez Becerra - 20251020047

Professor: Carlos Andres Sierra Virguez

Universidad Distrital Francisco Jose de Caldas

Subject: OOP

WORKSHOP #3

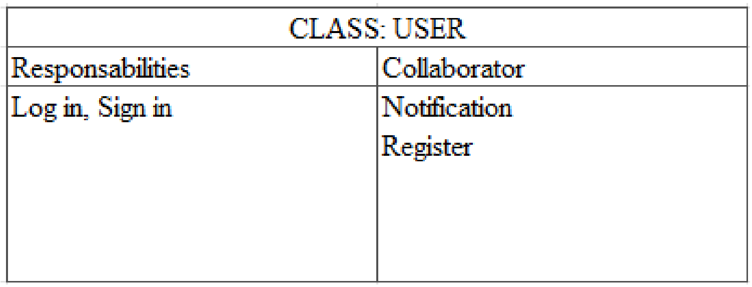
Revisiting Requirements & Design

Modifications to Requirements and User Stories

Both the requirements and the user stories remained unchanged, since when implementing the SOLID principles, the classes were restructured, but no functionalities were added or removed. The changes focused on the code structure.

Modifications to CRC Cards

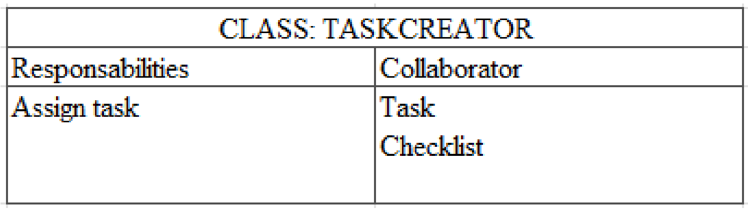
A distribution of the attributes of the classes that contained more than one was carried out, meaning that the responsibilities were separated, going from 3 parent classes to 7 independent classes.



CLASS: USER	
Responsabilities	Collaborator
Log in, Sign in	Notification Register

Figure 1

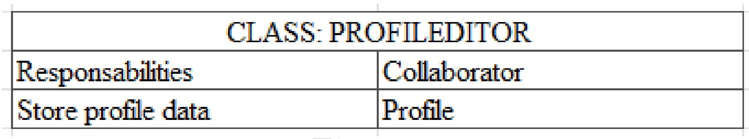
CRC Card User



CLASS: TASKCREATOR	
Responsabilities	Collaborator
Assign task	Task Checklist

Figure 2

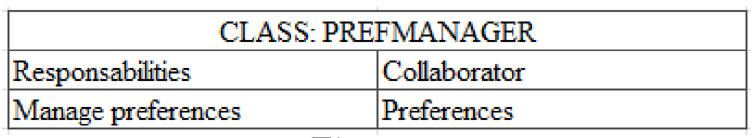
CRC Card TaskCreator



CLASS: PROFILEDITOR	
Responsabilities	Collaborator
Store profile data	Profile

Figure 3

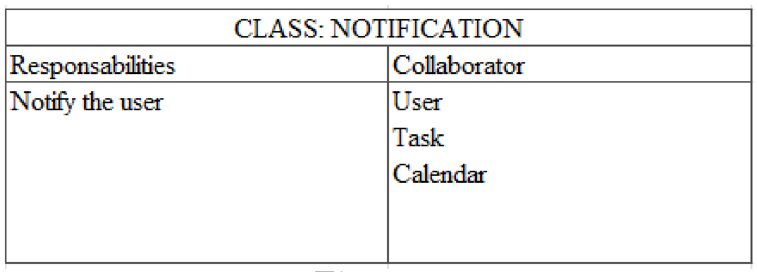
CRC Card ProfileEditor



CLASS: PREFMANAGER	
Responsabilities	Collaborator
Manage preferences	Preferences

Figure 4

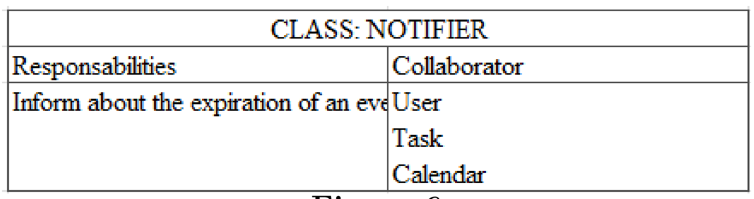
CRC Card PrefManager



CLASS: NOTIFICATION	
Responsabilities	Collaborator
Notify the user	User
	Task
	Calendar

Figure 5

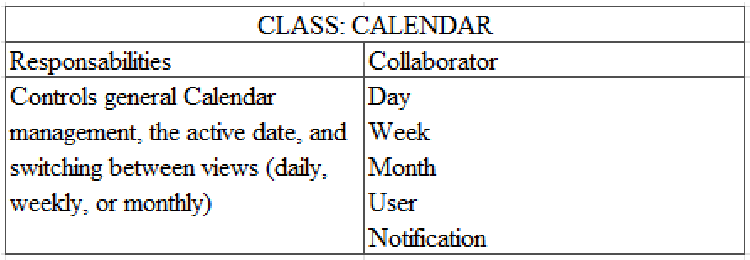
CRC Card Notification



CLASS: NOTIFIER	
Responsabilities	Collaborator
Inform about the expiration of an eve	User
	Task
	Calendar

Figure 6

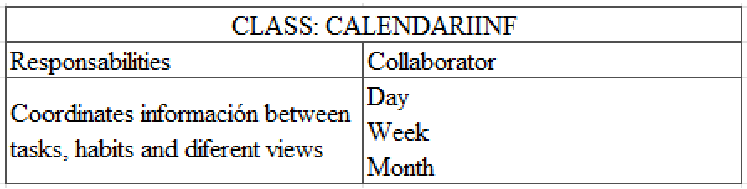
CRC Card Notifier



CLASS: CALENDAR	
Responsabilities	Collaborator
Controls general Calendar management, the active date, and switching between views (daily, weekly, or monthly)	Day Week Month User Notification

Figure 7

CRC Card Calendar



CLASS: CALENDARIINF	
Responsabilities	Collaborator
Coordinates información between tasks, habits and diferent views	Day Week Month

Figure 8

CRC Card CalendarInf

User responsibilities were divided into User, TaskCreator, ProfileEditor, and PrefManager; Notification responsibilities were divided into Notification and Notifier; and Calendar responsibilities were divided into Calendar and CalendarInf.

Enhanced UML Diagrams

In the implementation of the SOLID principles in the previous UML diagram, new classes were incorporated based on the changes proposed in the CRC Cards. To comply with the Single Responsibility Principle, functions were divided into more specific classes. In addition, new interfaces were added to ensure compliance with the Interface Segregation and Dependency Inversion principles, allowing responsibilities to be clearly separated and unnecessary dependencies between classes to be avoided.

Among the new interfaces, ProgressCalculator stands out, defining the behavior for calculating daily, weekly, and monthly progress, and used by the classes responsible for storing these values. The Assignable and Programmable interfaces were also added, implemented by the Task, Habit, and Event classes, since they share the functionality of assigning a duration and marking activities as completed. Finally, the CalendarInf interface was introduced to define the behavior for displaying activities, and it is implemented by the different calendar view classes.

Lastly, the relationships between classes were corrected to better reflect the new system structure and ensure full compliance with the SOLID principles.

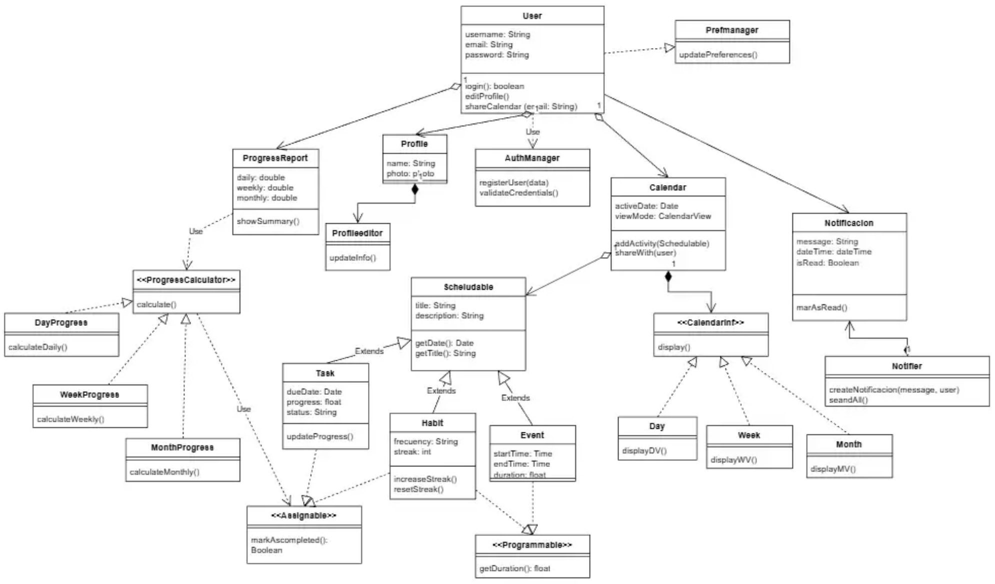


Figure 9

SOLID UML Diagram

SOLID-Focused Implementation

When implementing the SOLID principles, it is necessary to introduce new classes (as mentioned in the UML diagram and CRC Cards) to follow these principles, improve code maintainability, and avoid overloading responsibilities in existing classes. Additionally, applying these principles facilitates readability, modification, and scalability of the program in the future. Below is how each principle has been implemented:

Single Responsibility

In the previous design, the User, Notification, and Calendar classes were the only ones that did not comply with this principle. User was divided into four new classes:

1. User: handles registration and login.
2. TaskCreator: assigns tasks to the corresponding classes.
3. ProfileEditor: stores profile information.
4. PrefManager: modifies profile information.

Notification was divided into:

1.Notification: notifies users.

2.Notifier: informs about the status of events.

Calendar was divided into:

1.Calendar: manages the general content of the calendar.

2.CalendarInf: coordinates information between tasks, habits, and the different calendar views.

Open/Closed

Classes were organized to allow adding new functionality without modifying existing ones. The abstract class Schedulable enables the creation of new types of elements (such as a Meeting) simply by extending it. The same applies to calendar views (DayView, WeekView, MonthView), which share the same base but display different information depending on the view type.

Liskov Substitution

Derived classes can replace base classes without altering the expected behavior. For example, Task, Event, and Habit can replace the Schedulable class, as they represent variations of the base functionality rather than creating incompatible behaviors.

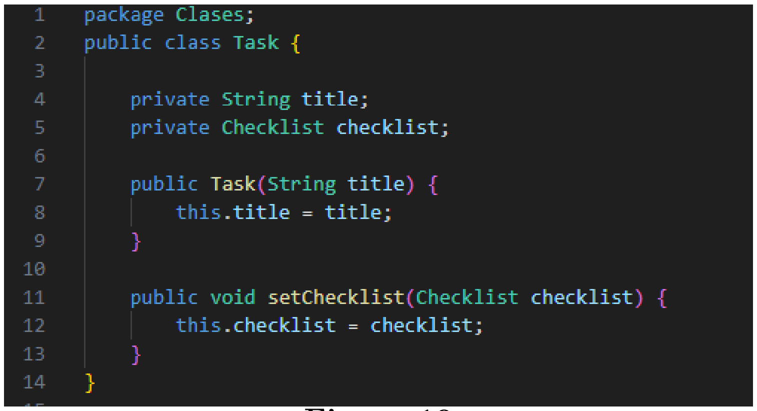
Interface Segregation

As the code evolves, interfaces are divided so that each class implements only those it actually needs. This prevents unnecessary implementations and allows new, specific interfaces to emerge according to each class's functionality.

Dependency Inversion

The system's most important functions (such as login or event creation) depend on interfaces rather than unnecessary concrete classes. This ensures greater flexibility and decoupling, making future changes easier without affecting the overall structure of the code.

Work in Progress Code & Documentation

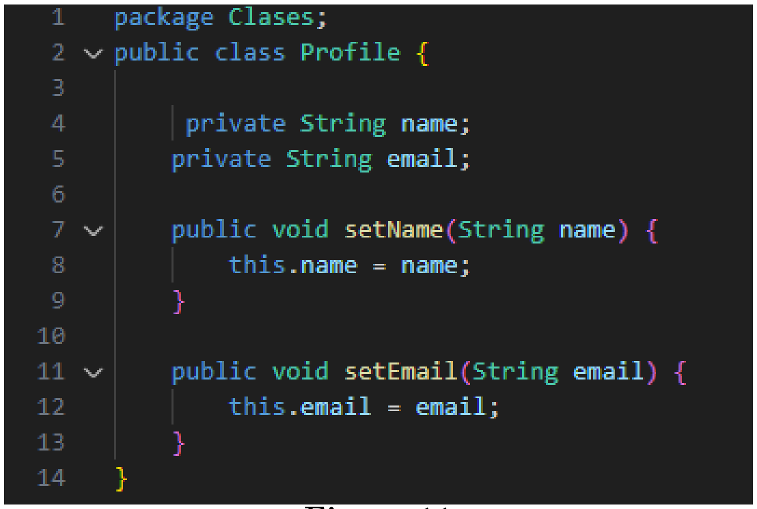


```java
package Clases;
public class Task {

    private String title;
    private Checklist checklist;

    public Task(String title) {
        this.title = title;
    }

    public void setChecklist(Checklist checklist) {
        this.checklist = checklist;
    }
}
```

Figure 10

Class Task

This class represent a Task. Previously, it was part of User, but it was modified to comply with the SOLID principles.

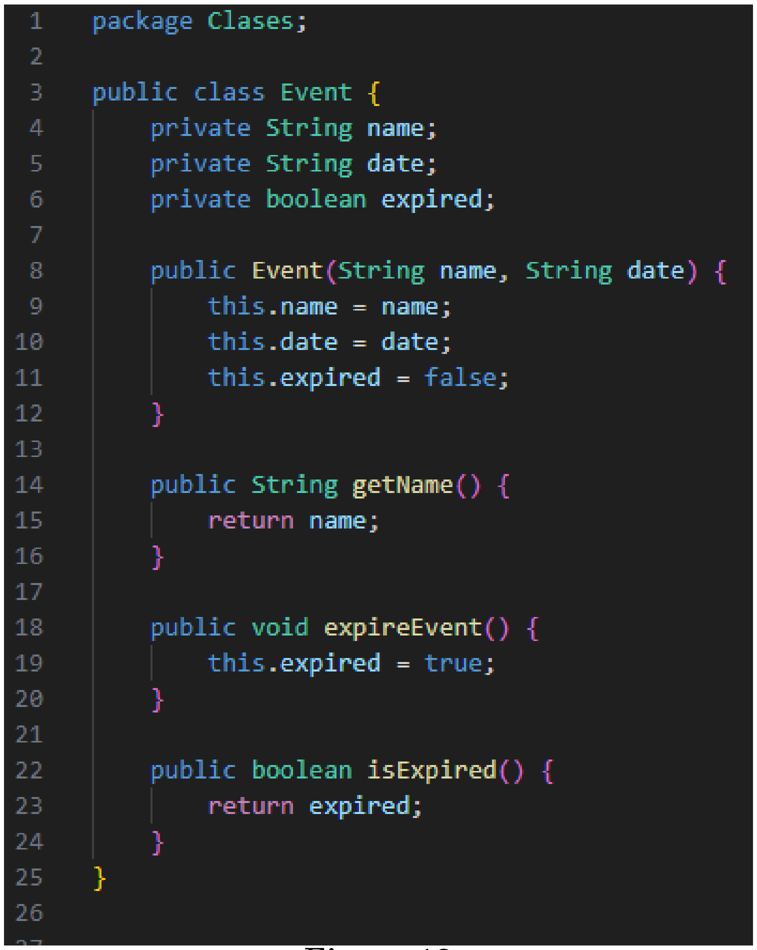


```java
package Clases;
public class Profile {

     private String name;
    private String email;

    public void setName(String name) {
        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Figure 11

Class Profile

This class is responsible for storing user information. Previously, it was part of User, but it was modified to comply with SOLID principles.

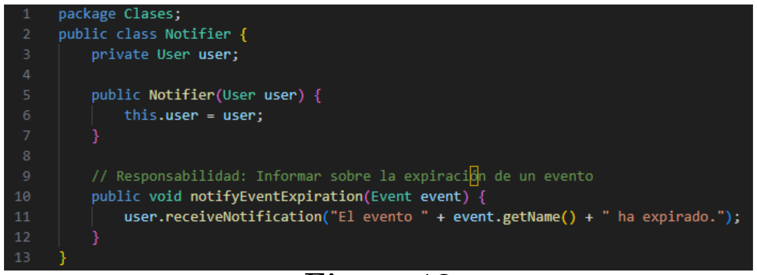


```java
package Clases;

public class Event {
    private String name;
    private String date;
    private boolean expired;

    public Event(String name, String date) {
        this.name = name;
        this.date = date;
        this.expired = false;
    }

    public String getName() {
        return name;
    }

    public void expireEvent() {
        this.expired = true;
    }

    public boolean isExpired() {
        return expired;
    }
}
```

Figure 12

Class Event

This class represents an event and will also mark when an event has expired.

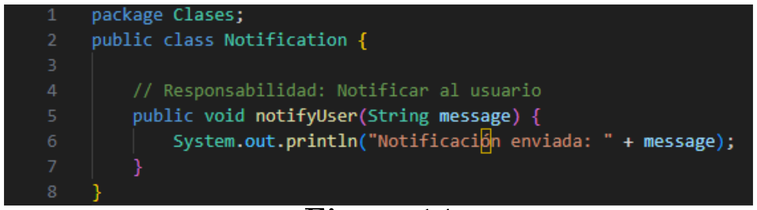


```java
package Clases;
public class Notifier {
    private User user;

    public Notifier(User user) {
        this.user = user;
    }

    // Responsabilidad: Informar sobre la expiración de un evento
    public void notifyEventExpiration(Event event) {
        user.receiveNotification("El evento " + event.getName() + " ha expirado.");
    }
}
```

Figure 13

Class Notifier

This class is responsible for notifying the user of the status of an event. It was previously located in Notification, but was moved to comply with the principle of single responsibility.



```java
package Clases;
public class Notification {

    // Responsabilidad: Notificar al usuario
    public void notifyUser(String message) {
        System.out.println("Notificación enviada: " + message);
    }
}
```

Figure 14

Class Notification

This class is responsible for notifying the user. Previously, the responsibility of the Notifier class toward this class was modified to comply with the

Brief Reflection

During the development of Workshop #3, we focused on applying the SOLID principles within the Object-Oriented Programming concepts previously used in the design and creation of the application. Throughout the process, we faced several challenges related to the proper implementation of the SOLID principles, particularly in the use of interfaces and in understanding the SRP principle, which states that each class should have a single responsibility. At this stage, we struggled to determine how far we should divide or distribute the methods among the classes. Overall, this workshop helped us understand the importance of SOLID principles and their impact on software development, strengthening our skills in design, teamwork, and logical thinking within the context of object-oriented programming.