# Diseño Gestión de Tareas

## Introducción

Este documento describe un sistema simple de gestión de tareas utilizando programación orientada a objetos (OOP) en Java. El sistema permite crear proyectos, añadir tareas, asignar tareas a usuarios y cambiar el estado de las tareas. Además, se implementan interfaces para promover la abstracción y la reutilización del código. Se han aplicado los principios SOLID para garantizar que el sistema sea flexible, mantenible y escalable.

## Clases y Métodos

### 1. ****Interfaz IProject****

interface IProject {

void addTask(Task task);

String getName();

}

#### Descripción

La interfaz IProject define el contrato para cualquier clase que represente un proyecto. Contiene los siguientes métodos:

* **addTask(Task task)**: Permite agregar una tarea al proyecto.
* **getName()**: Devuelve el nombre del proyecto.

### 2. ****Clase Project****

import java.util.ArrayList;

import java.util.List;

class Project implements IProject {

private String name;

private List<Task> tasks;

public Project(String name) {

this.name = name;

this.tasks = new ArrayList<>();

}

@Override

public void addTask(Task task) {

tasks.add(task);

}

@Override

public String getName() {

return name;

}

public List<Task> getTasks() {

return tasks;

}

}

#### Descripción

La clase Project implementa la interfaz IProject. Contiene un nombre y una lista de tareas. Los métodos incluidos son:

* **Constructor Project(String name)**: Inicializa el proyecto con un nombre y una lista vacía de tareas.
* **addTask(Task task)**: Implementa el método de la interfaz para agregar una tarea.
* **getName()**: Devuelve el nombre del proyecto.
* **getTasks()**: Permite acceder a la lista de tareas del proyecto.

### 3. ****Interfaz ITask****

interface ITask {

void assignUser(User user);

void completeTask();

String getStatus();

void updateStatus(boolean completed);

}

#### Descripción

La interfaz ITask define las operaciones que pueden realizarse sobre una tarea. Sus métodos son:

* **assignUser(User user)**: Asigna un usuario a la tarea.
* **completeTask()**: Marca la tarea como completada.
* **getStatus()**: Devuelve el estado actual de la tarea.
* **updateStatus(boolean completed)**: Actualiza el estado de la tarea.

### 4. ****Clase Task****

class Task implements ITask {

private String name;

private User assignedUser;

private boolean completed;

public Task(String name) {

this.name = name;

this.completed = false;

}

@Override

public void assignUser(User user) {

this.assignedUser = user;

}

@Override

public void completeTask() {

this.completed = true;

}

@Override

public String getStatus() {

return completed ? "Completed" : "Pending";

}

@Override

public String toString() {

return "Tarea: " + name + ", Estado: " + getStatus();

}

@Override

public void updateStatus(boolean completed) {

this.completed = completed;

}

public User getAssignedUser() {

return assignedUser;

}

}

#### Descripción

La clase Task implementa la interfaz ITask. Cada tarea tiene un nombre, un usuario asignado y un estado de completado. Los métodos son:

* **Constructor Task(String name)**: Inicializa la tarea con un nombre y el estado como no completada.
* **assignUser(User user)**: Asigna un usuario a la tarea.
* **completeTask()**: Cambia el estado de la tarea a completada.
* **getStatus()**: Devuelve el estado de la tarea.
* **toString()**: Representa la tarea como una cadena de texto.
* **updateStatus(boolean completed)**: Actualiza el estado de completado de la tarea.
* **getAssignedUser()**: Devuelve el usuario asignado a la tarea.

### 5. ****Interfaz ITaskManager****

import java.util.List;

interface ITaskManager {

void addTask(Task task);

void assignTaskToUser(Task task, User user);

List<Task> getTasks();

}

#### Descripción

La interfaz ITaskManager define el comportamiento esperado para un administrador de tareas. Sus métodos son:

* **addTask(Task task)**: Añade una tarea al gestor.
* **assignTaskToUser(Task task, User user)**: Asigna una tarea a un usuario.
* **getTasks()**: Devuelve una lista de tareas gestionadas.

### 6. ****Clase TaskManager****

import java.util.ArrayList;

import java.util.List;

class TaskManager implements ITaskManager {

private List<Task> tasks;

public TaskManager() {

tasks = new ArrayList<>();

}

@Override

public void addTask(Task task) {

tasks.add(task);

}

@Override

public void assignTaskToUser(Task task, User user) {

task.assignUser(user);

}

@Override

public List<Task> getTasks() {

return tasks;

}

}

#### Descripción

La clase TaskManager implementa la interfaz ITaskManager. Se encarga de gestionar las tareas. Sus métodos son:

* **Constructor TaskManager()**: Inicializa una lista vacía de tareas.
* **addTask(Task task)**: Implementa el método para agregar una tarea.
* **assignTaskToUser(Task task, User user)**: Asigna un usuario a una tarea.
* **getTasks()**: Devuelve la lista de tareas.

### 7. ****Interfaz IUser****

interface IUser {

String getName();

}

#### Descripción

La interfaz IUser define el comportamiento de un usuario. Tiene un único método:

* **getName()**: Devuelve el nombre del usuario.

### 8. ****Clase User****

class User implements IUser {

private String name;

public User(String name) {

this.name = name;

}

@Override

public String getName() {

return name;

}

}

#### Descripción

La clase User implementa la interfaz IUser. Representa a un usuario en el sistema. Sus métodos son:

* **Constructor User(String name)**: Inicializa el usuario con un nombre.
* **getName()**: Devuelve el nombre del usuario.

# 9. Clase TaskFactory

**Descripción**

La clase TaskFactory es responsable de crear instancias de la clase Task. Utiliza el patrón de diseño de fábrica para facilitar la creación de objetos sin necesidad de especificar la clase exacta de los objetos que se están creando.

**Métodos**

**Task createTask(String name)**

* **Descripción**: Crea una nueva tarea con el nombre especificado.
* **Parámetros**:
  + name: El nombre de la tarea a crear.
* **Retorno**: Una nueva instancia de Task con el nombre proporcionado.

# 10. Clase UserFactory

**Descripción**

La clase UserFactory es responsable de crear instancias de la clase User. Utiliza el patrón de diseño de fábrica para facilitar la creación de objetos sin necesidad de especificar la clase exacta de los objetos que se están creando.

**Métodos**

**User createUser(String name)**

* **Descripción**: Crea un nuevo usuario con el nombre especificado.
* **Parámetros**:
  + name: El nombre del usuario a crear.
* **Retorno**: Una nueva instancia de User con el nombre proporcionado.

## Implementación de los Principios SOLID

### S - ****Single Responsibility Principle (SRP)****

Cada clase en el sistema tiene una única responsabilidad. Por ejemplo, la clase Task se encarga únicamente de representar una tarea y sus operaciones, mientras que la clase TaskManager gestiona las tareas. Esto facilita la mantenibilidad y comprensión del código.

### O - ****Open/Closed Principle (OCP)****

Las clases están diseñadas para ser abiertas a la extensión pero cerradas a la modificación. Por ejemplo, se pueden añadir nuevas implementaciones de las interfaces ITask, IProject y ITaskManager sin modificar el código existente, lo que permite añadir funcionalidades adicionales.

### L - ****Liskov Substitution Principle (LSP)****

Cualquier clase que implemente las interfaces puede ser utilizada sin problemas en el sistema. Por ejemplo, si se crea una nueva clase que implementa IUser, puede ser utilizada donde se espera un User sin afectar el funcionamiento del sistema.

### I - ****Interface Segregation Principle (ISP)****

Las interfaces están diseñadas de tal manera que no obligan a las clases a implementar métodos que no utilizarán. Por ejemplo, la interfaz ITask define solo métodos relevantes para las tareas, sin incluir métodos innecesarios.

### D - ****Dependency Inversion Principle (DIP)****

El sistema depende de abstracciones (interfaces) en lugar de clases concretas. Esto se refleja en el uso de interfaces como IProject, ITask, ITaskManager, y IUser, lo que permite desacoplar el código y facilitar la prueba y la extensibilidad.

Pruebas realizadas

Menu de opciones :

Se agrega una tarea:

Texto

Descripción generada automáticamente con confianza media

Se asigna una tarea:

Texto

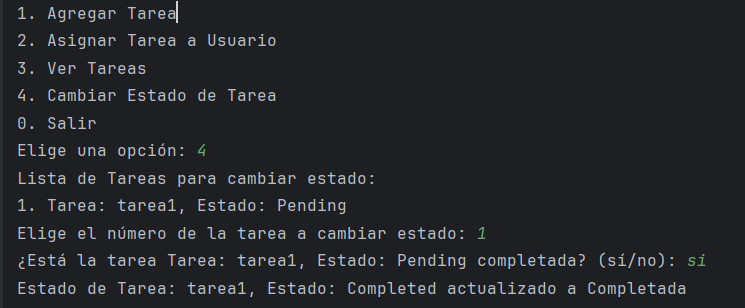
Descripción generada automáticamente

Se consula la tarea:

Texto

Descripción generada automáticamente

Cambio de estado de tarea:



## Conclusión

El sistema de gestión de tareas presentado es un ejemplo de cómo aplicar los principios de diseño orientado a objetos, en particular los principios SOLID, para crear un software mantenible y escalable. La clara separación de responsabilidades y la utilización de interfaces permiten que el sistema se adapte a futuros requerimientos sin la necesidad de realizar cambios drásticos en el código existente.