Objetivos Específicos

1. Crear modelos que representen la estructura y el comportamiento de un sistema de software, a partir de las especificaciones de los requerimientos del software usando patrones de diseño y aplicando los principios SOLID.

Resultado de Aprendizaje

1. Diseñar, implementar y evaluar una solución basada en computación para cumplir con un conjunto dado de requisitos de computación en el contexto de la disciplina del programa.
2. Habilidad para aplicar teoría de ciencias computacionales y fundamentos de desarrollo de software para producir soluciones basadas en computación.

Descripción del taller

Desarrollar individualmente una solución que modele un sistema informático utilizando diagramas de clases y de secuencia UML, además de aplicar los principio de diseño SOLID y los patrones de diseño creacionales y estructurales clásicos.

Sección A

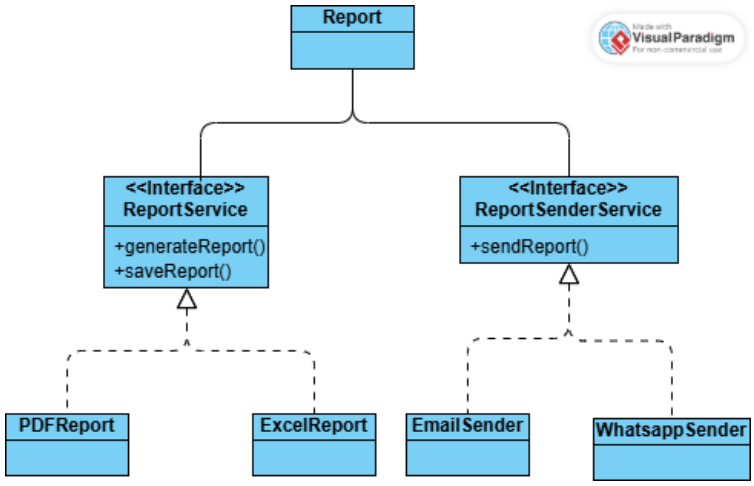
Considere el siguiente fragmento de código que pertenece a un sistema de generación y distribución de reportes. El sistema permite generar reportes en distintos formatos, almacenarlos, y enviarlos a través de diferentes medios.

```
public class ReportService {
    public void generateReport(String type) {
        if (type.equals("PDF")) {
            System.out.println("Generando reporte en PDF");
        } else if (type.equals("EXCEL")) {
            System.out.println("Generando reporte en Excel");
        }
    }

    public void saveReport(String type) {
        if (type.equals("PDF")) {
            System.out.println("Guardando PDF en disco");
        } else if (type.equals("EXCEL")) {
            System.out.println("Guardando Excel en disco");
        }
    }

    public void sendReport(String type, String method) {
```

```
        if (method.equals("EMAIL")) {
            System.out.println("Enviando " + type + " por correo electrónico");
        } else if (method.equals("WHATSAPP")) {
            System.out.println("Enviando " + type + " por WhatsApp");
        }
    }
}
```

Instrucciones:

1. Identifique las violaciones a los principios SOLID que se encuentran en este código. Justifique su respuesta.
   Single Resposability Principle: el método enviar reportes esta mejor ubicado en una clase aparte ya que no tiene relación con el resto de métodos, dándole cargas de mas a la clase ReportServices
   Open Close Principle: No está abierto extenciones, ya que está limitado solamente a los formatos que se detallan en las estructuras de control, por lo tanto si se intenta agregar otro formato se violaría el hecho de estar cerrado a modificaciones, ya que se debe modificar el código para agregar otro formato/Servicio
   Interface Segregation Principle: Es mejor tener varias interfaces especificas que una sola muy general, esto aplicaría para ReportService que se permite crear 2 interfaces especificas, una siendo ReportSender y la base de ReportService
   [10%]
2. Rediseñe utilizando clases y/o interfaces nuevas que permitan aplicar los principios SOLID. Agregue el diagrama de clases correspondiente aplicando SOLID. [15%]
3. Genere el mínimo código JAVA de su diagrama de clases. No es necesario incluir código de envío real, solo la estructura de clases e interfaces con métodos vacíos o con System.out.println(...). [10%]



Sección B

A partir del siguiente caso de uso y del diagrama de clases proporcionado, elabore un diagrama de secuencia UML para la operación createTask().

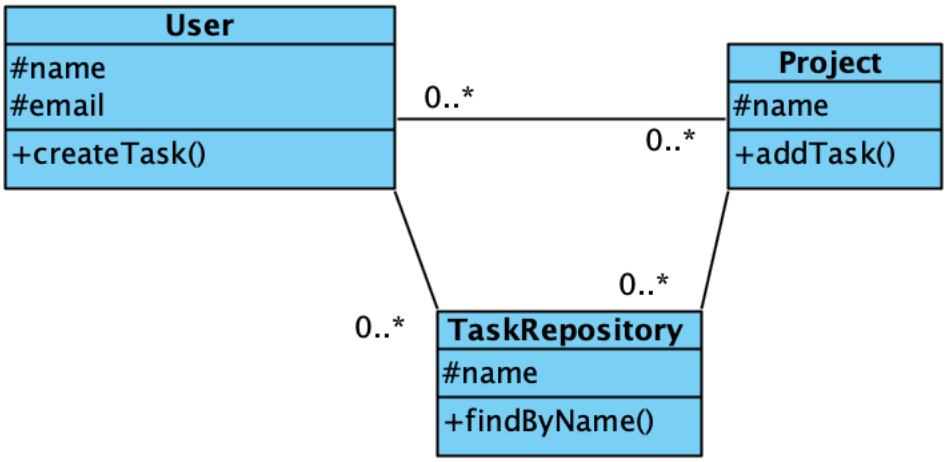
Caso de Uso: Registrar tarea en proyecto
Actor principal: Usuario (User)
Descripción breve: Un usuario crea una nueva tarea dentro de un proyecto existente. El sistema valida que el nombre de la tarea no esté duplicado, asocia la tarea al proyecto y la guarda en el repositorio. Finalmente, se notifica a los usuarios asignados.
Flujo principal de eventos:

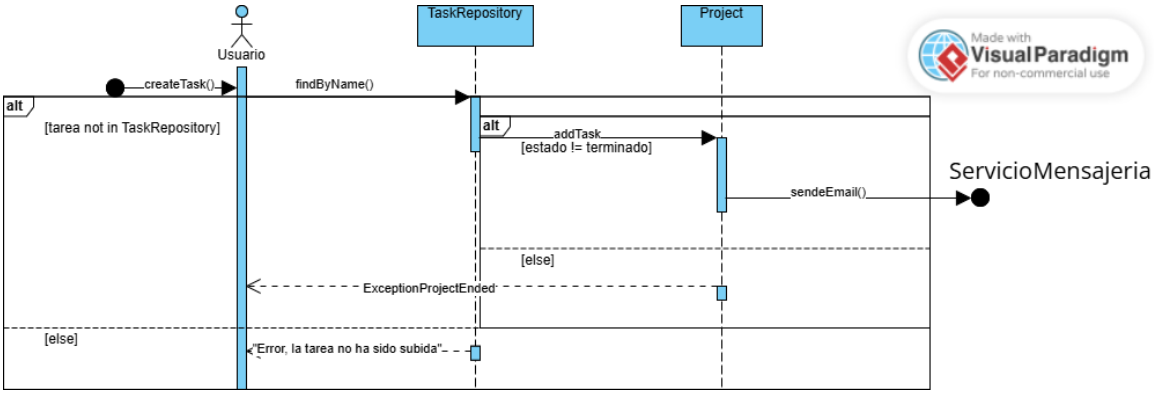
1. El usuario accede al proyecto en el que desea registrar una nueva tarea.
2. El usuario crea una tarea proporcionando su nombre.
3. El sistema consulta en el TaskRepository si ya existe una tarea con ese nombre en el proyecto.
4. Si no existe, el sistema guarda la nueva tarea en el TaskRepository.
5. El sistema asocia la tarea al Project.
6. El sistema notifica por email al usuario que se ha registrados la tarea.

Flujos alternativos:
- 3a. Tarea duplicada: Si ya existe la tarea con el mismo nombre en el proyecto, el sistema muestra un mensaje de error y cancela la operación.
- 5a. Proyecto terminado: Si el proyecto tiene estado terminado, se lanza una excepción y no se registra la tarea.

Postcondiciones:
- La tarea queda registrada en el repositorio de tareas.
- La tarea queda asociada al proyecto correspondiente.
- Los usuarios asignados conocen su responsabilidad en la nueva tarea.

Diagrama de clases del sistema:





Realizar el diagrama de secuencias especificando:

1. Clases [05%]
2. Actores [05%]
3. Barras de activación [05%]
4. Mensajes (Síncronos, asíncronos, creación y retorno) [10%]
5. Bloques (Condicionales, Lazos) [10%]

Sección C

Descripción del sistema: Generador de Presentaciones Multiformato

Se desea construir un sistema llamado SlideComposer que permita a los usuarios crear presentaciones personalizadas, con múltiples secciones, temas, contenidos (texto, imágenes, videos) y estilos. Una vez construida la presentación, esta puede exportarse a diferentes formatos: PDF, PowerPoint (PPTX), HTML o Markdown.

Cada formato de exportación requiere una estructura distinta:

- PDF y PPTX se generan con librerías externas con interfaces incompatibles entre sí.
- HTML y Markdown se pueden construir directamente con cadenas y plantillas internas del sistema.

Además, se desea que la creación de una presentación sea flexible, permitiendo construirla paso a paso con opciones como:

- Agregar secciones con título.
- Agregar contenido multimedia.
- Cambiar el estilo visual.

Se requiere:

1. Modele un diagrama de clases UML que represente su solución, destacando los patrones utilizados.
2. Aplique el patrón 1 para permitir la construcción paso a paso de una presentación compleja y personalizable. [10%]
3. Aplique el patrón 2 para integrar diferentes librerías de exportación (por ejemplo, una clase externa PPTXExporter que no puede ser modificada). [10%]
4. Justifique brevemente por qué su diseño aplica correctamente los patrones. [10%]

Mi diseño permite exportar las presentaciones a cualquier formato que se encuentre dentro del sistema, más un adapter que permite librerías externas con interfaces completamente distintas incompatibles. Permite personalizar la presentación de forma flexible con distintas opciones dadas por un builder