Diseño y Programación Orientada a Objetos – Sección 6

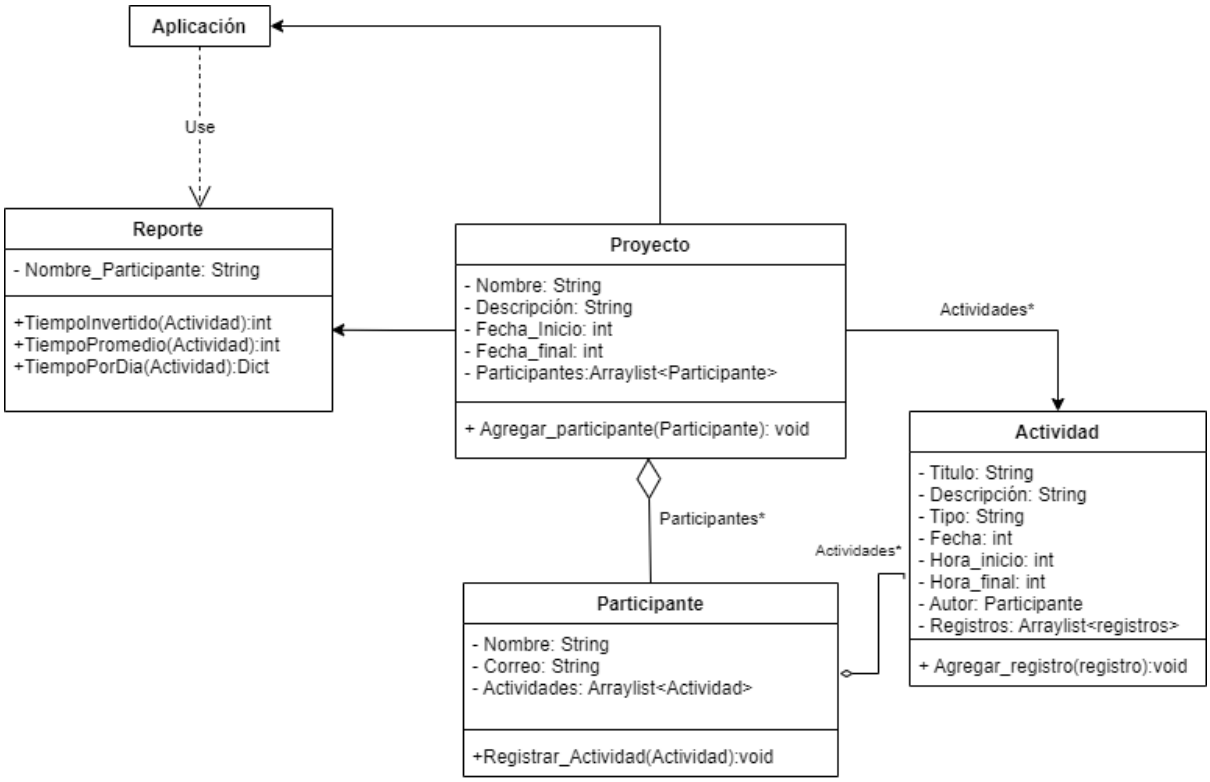
Jerónimo Vargas Rendon - 202113305

Nicolas Camargo Prieto – 202020782

Documento de Diseño

Análisis

Modelo de Dominio (UML):



Descripción de Requerimientos Funcionales:

HISTORIAS DE USUARIO

1. Como participante de un proyecto, quiero poder tener un sistema que registre todo el proceso de desarrollo de dicho proyecto, con el fin de conocer los miembros que participaron e información de las diferentes tareas que se efectuaron los miembros del grupo.

- Registrar en un sistema información del proyecto.
- Ver los miembros del proyecto.
- Ver registro de actividades.
- Ver información calculada de los datos.

2. Como dueño del proyecto creado, quiero poder registrar nuevos participantes ya que inicialmente soy el único, mientras que es muy probable que un proyecto se conforme de varios participantes que aportan al desarrollo de este mismo.

- Registrar los miembros del proyecto (Crear un participante).
- Registrar nombre y correo del participante.

3. Como participante del proyecto, deseo poder registrar en el sistema las actividades que voy realizando en el proyecto, para dejar soporte de la tarea que realicé, con la fecha en la que se realizó, información que la caracteriza y tiempos en el cual se desarrolló.

4. Como participante del proyecto, quiero poder hacer varios registros o commits en una misma tarea, debido a que la actividad a realizar puede que tarde días en realizarse.

5. Como participante del proyecto, quiero poder cronometrar el tiempo que tardo en realizar una actividad del proyecto, además deseo poder detener el cronometró en caso de que deba interrumpir el desarrollo de la actividad.

6. Como dueño del proyecto, quiero poder recibir un reporte que informa de las actividades realizadas por cada miembro y diferentes características calculadas.

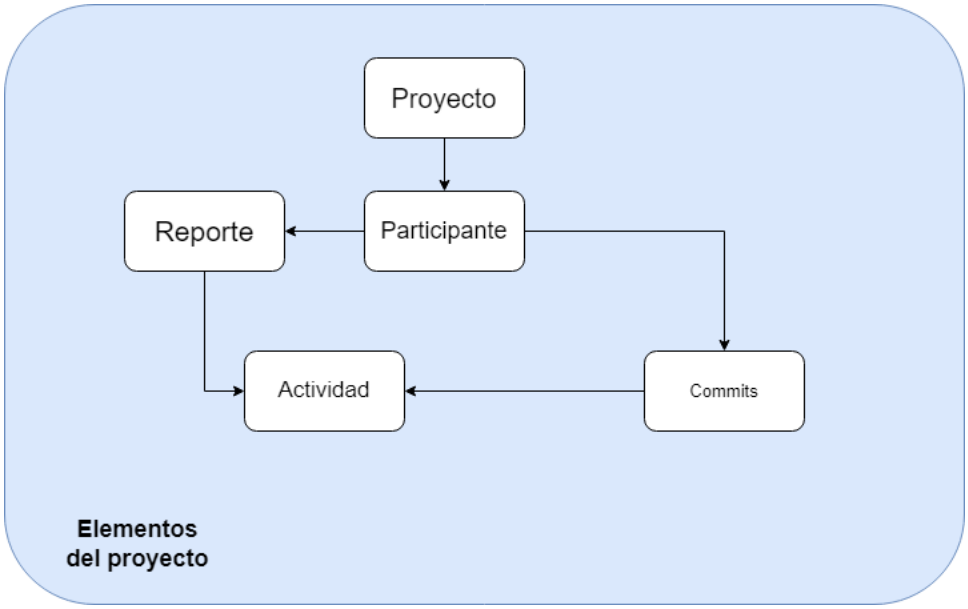
- Ver total actividades.
- Ver Tiempo total invertido.
- Ver Tiempo promedio por actividad.

Restricciones:

Aunque el diseño del proyecto incluye la mayoría de los aspectos esenciales y requerimientos funcionales, también este cuenta con unas cuentas restricciones. Por un lado, el proyecto solo puede ser realizado por dos personas, quienes deben tener en cuenta los cambios realizados por el otro. Se debe usar eclipse (Java) para su realización, y de la misma forma, todos los cambios deben ser subidos a GitHub. Por otro lado, la modificación de datos en el registro puede generar conflictos con la fecha y hora en la que se realizaron las actividades. A sí mismo, establecer en el diseño varias tareas con el mismo nombre, puede suponer conflictos a la hora de guardar la información. Por último, el tipo de actividad que debe ser definida (Documentación, Implementación, Pruebas, etc.) deben ser seleccionadas previamente, pues hacen parte de un arraylist dentro de la clase proyecto, que es importante para interactuar con el resto de las clases.

Diseño

Agrupación de Elementos:



Estereotipos (roles):

- **Proyecto:** Structurer, pues es responsable de mantener la estructura, información y ciertos comandos que se deben hacer
- **Participante:** Controller, ya que es el principal responsable del proyecto, asi como, el que se encarga de las tareas como hacer commits o generar un reporte en base a una actividad.
- **Reporte:** Information holder, tiene que almacenar los reportes.
- **Actividad:** Structurer, mantiene información sobre las actividades, commits y genera los reportes.
- **Commits:** Service provider, permite al miembro o participante guardar informacion sobre una actividad.

Ahora el proceso de diseño iterativo:

- Por cada iteración, los elementos añadidos serán resaltados con un color.
- Se enumeran los objetos y las responsabilidades.
- Cada iteración cuenta con los objetos, las responsabilidades y las colaboraciones entre ellos denotadas en una lista que indica los números de los objetos que están involucrados en la colaboración.
- En total son 3 iteraciones (contando la final).

Primera Iteración - []

ROLES

1. Proyecto
2. Participante
3. Reporte
4. Actividad
5. Commit

RESPONSABILIDADES

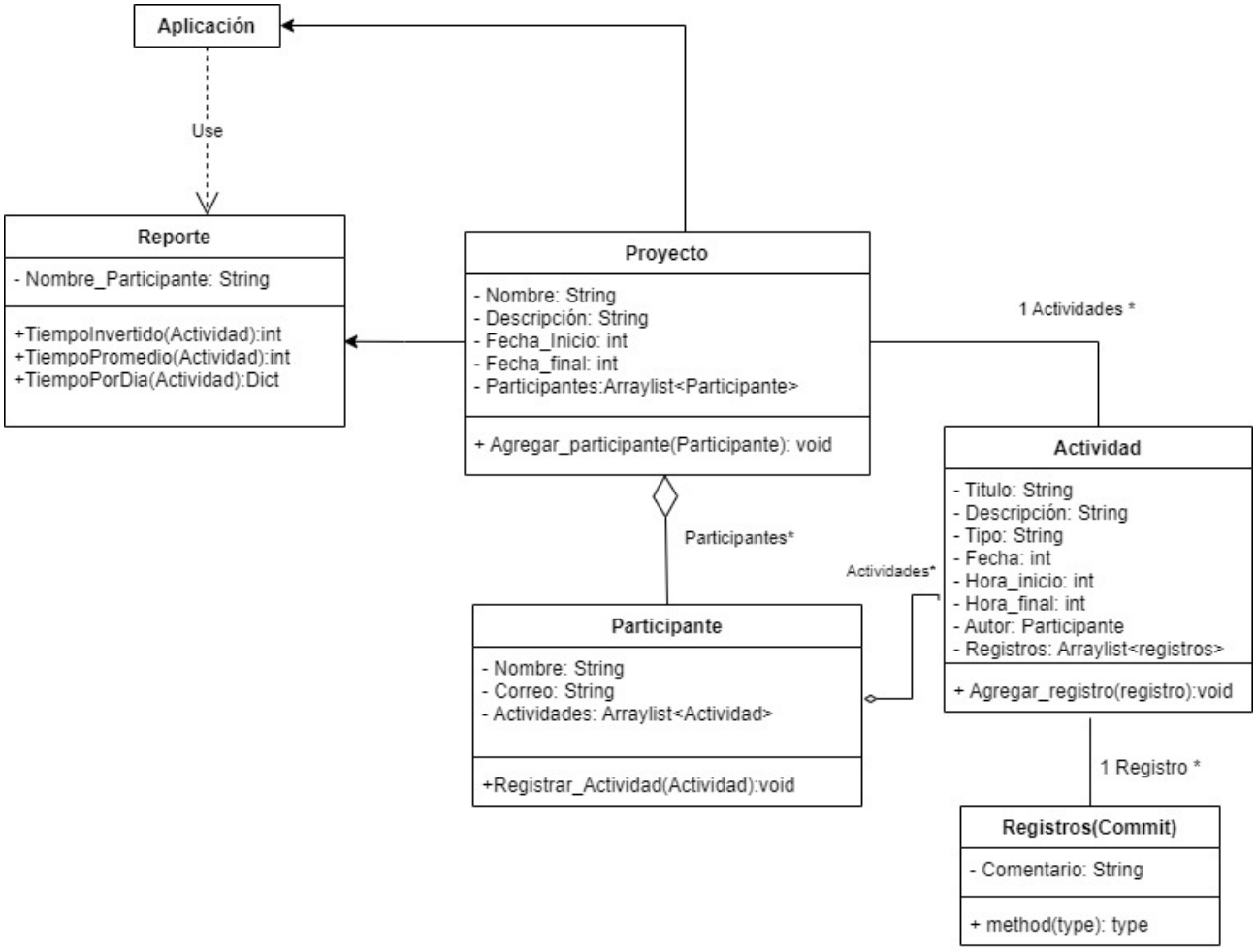
1. Crear Proyecto
2. Agregar miembros del proyecto
3. Eliminar miembros del proyecto
4. Registrar Actividad
5. Registrar Commit o registro en la Actividad
6. Realizar Reporte

COLABORACIONES

#Responsabilidad => Roles []

1. => [1,2]
2. => [1,2]
3. => [1,2]
4. => [1,2,4,5]
5. => [1,4,5]
6. => [1,3]

Diagrama #1 Iteración



Segunda Iteración - []

OBJETOS

1. Proyecto
2. Participante
3. Reporte
4. Actividad
5. Commit
6. Cronometro

RESPONSABILIDADES

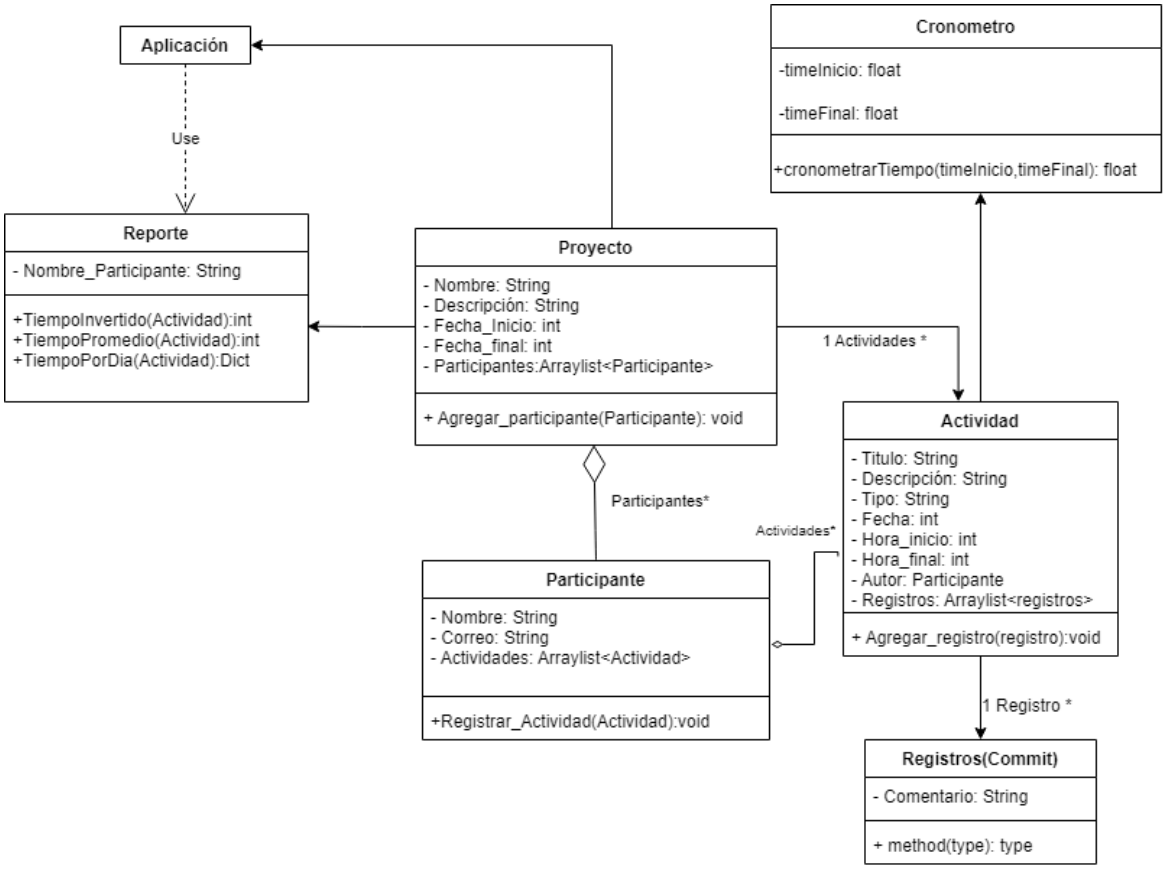
1. Crear Proyecto
2. Agregar miembros del proyecto
3. Eliminar miembros del proyecto
4. Registrar Actividad
5. Registrar Commit o registro en la Actividad
6. Realizar Reporte
7. Cronometrar y calcular el tiempo que tomo el usuario en realizar una actividad

COLABORACIONES

#Responsabilidad => Roles []

1. => [1,2]
2. => [1,2]
3. => [1,2]
4. => [1,2,4,5]
5. => [1,4,5]
6. => [1,3]
7. => [7,4]

Diagrama #2 Iteración



Justificación Segunda Iteración: Una vez planteada la primera iteración se volvió hacer un barrido sobre los requerimientos funcionales y responsabilidades y se identificó que hacía falta una responsabilidad, la cual era permitir que un miembro del proyecto pueda cronometrar lo que tarda realizando una actividad.

Tercera Iteración - []

OBJETOS

1. Proyecto
2. Participante
3. Reporte
4. Actividad
5. Commit
6. Cronometro
7. Consola

RESPONSABILIDADES

1. Crear Proyecto
2. Agregar miembros del proyecto
3. Eliminar miembros del proyecto
4. Registrar Actividad
5. Registrar Commit o registro en la Actividad

6. Realizar Reporte
7. Cronometrar y calcular el tiempo que tomo el usuario en realizar una actividad
8. Recibir informacion (input) y mostrar el output.

COLABORACIONES

#Responsabilidad => Roles []

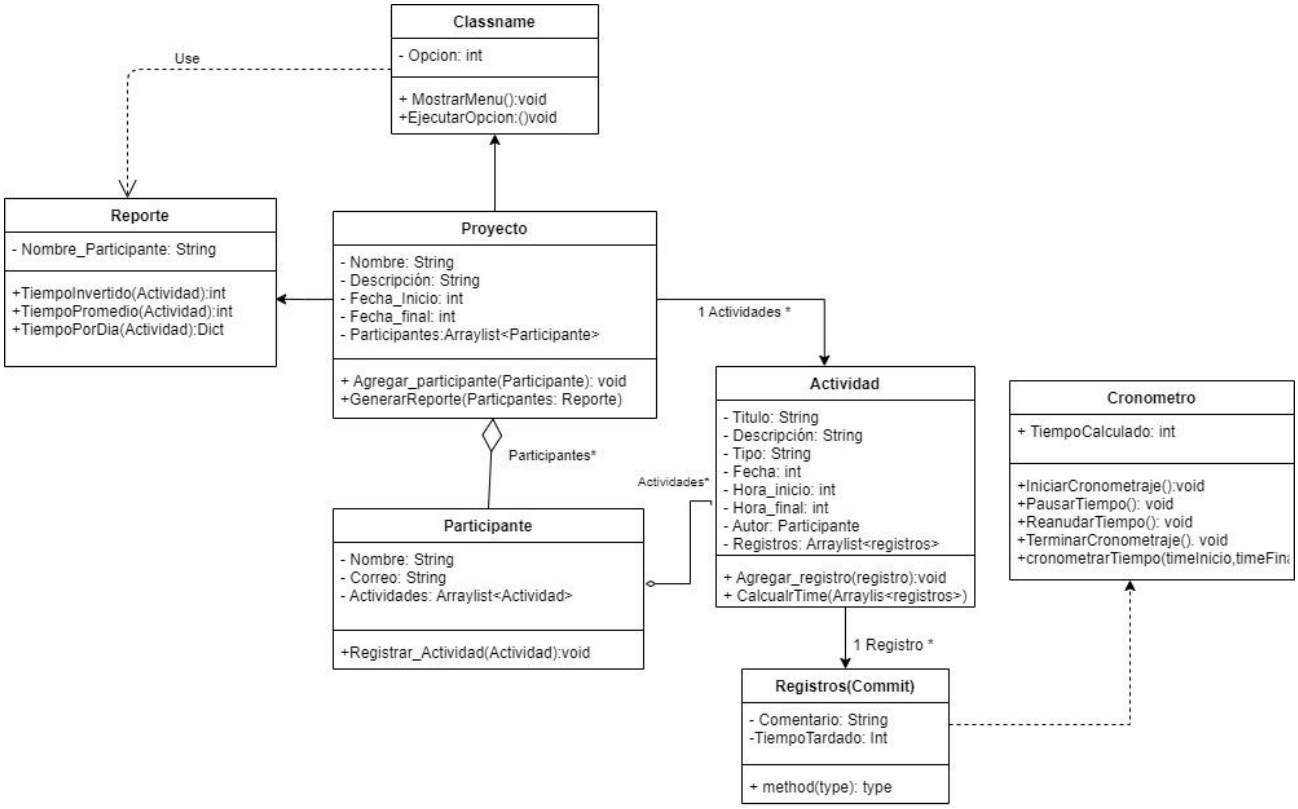
1. => [1,2]
2. => [1,2]
3. => [1,2]
4. => [1,2,4,5]
5. => [1,4,5]
6. => [1,3]
7. => [7,4]
8. => [8,4]

Justificación Tercera Iteración: Esta tercera iteración se debe a que faltaba un componente externo a la lógica de dominio, este componente es la consola que permite controlar toda la interacción con el Usuario mediante Inputs y Outputs para poder hacer que la lógica desarrollada pueda ser usable. Por otro lado, se detectaron que faltaba definir algunos métodos y atributos en algunos elementos especialmente el Cronometro y Actividad.

ESTEREOTIPOS:

- **Proyecto:** Structurer, pues es responsable de mantener la estructura, información y ciertos comandos que se deben hacer
- **Participante:** Controller, ya que es el principal responsable del proyecto, así como, el que se encarga de las tareas como hacer commits o generar un reporte en base a una actividad.
- **Reporte:** Information holder, tiene que almacenar los reportes.
- **Actividad:** Structurer, mantiene información sobre las actividades, commits y genera los reportes.
- **Commits:** Service provider, permite al miembro o participante guardar información sobre una actividad.
- **Consola:** Interfacer, permite al usuario introducir información y recibir resultados.
- **Cronometro:** Service provider, realiza el cálculo del tiempo de una actividad, almacena y despliega esta información.

Diagrama #3 Iteración



Cuarta Iteración (FINAL) - []

OBJETOS

1. Proyecto
2. Participante
3. Reporte
4. Actividad
5. Cronometro
6. Consola

RESPONSABILIDADES

1. Crear Proyecto
2. Modificar Fecha Final del Proyecto
3. Agregar miembros del proyecto
4. Eliminar miembros del proyecto
5. Mostrar Información del Proyecto
6. Registrar Actividad
7. Realizar Reporte
8. Cronometrar Actividad y agregarla
9. Recibir información (input) y mostrar el output.

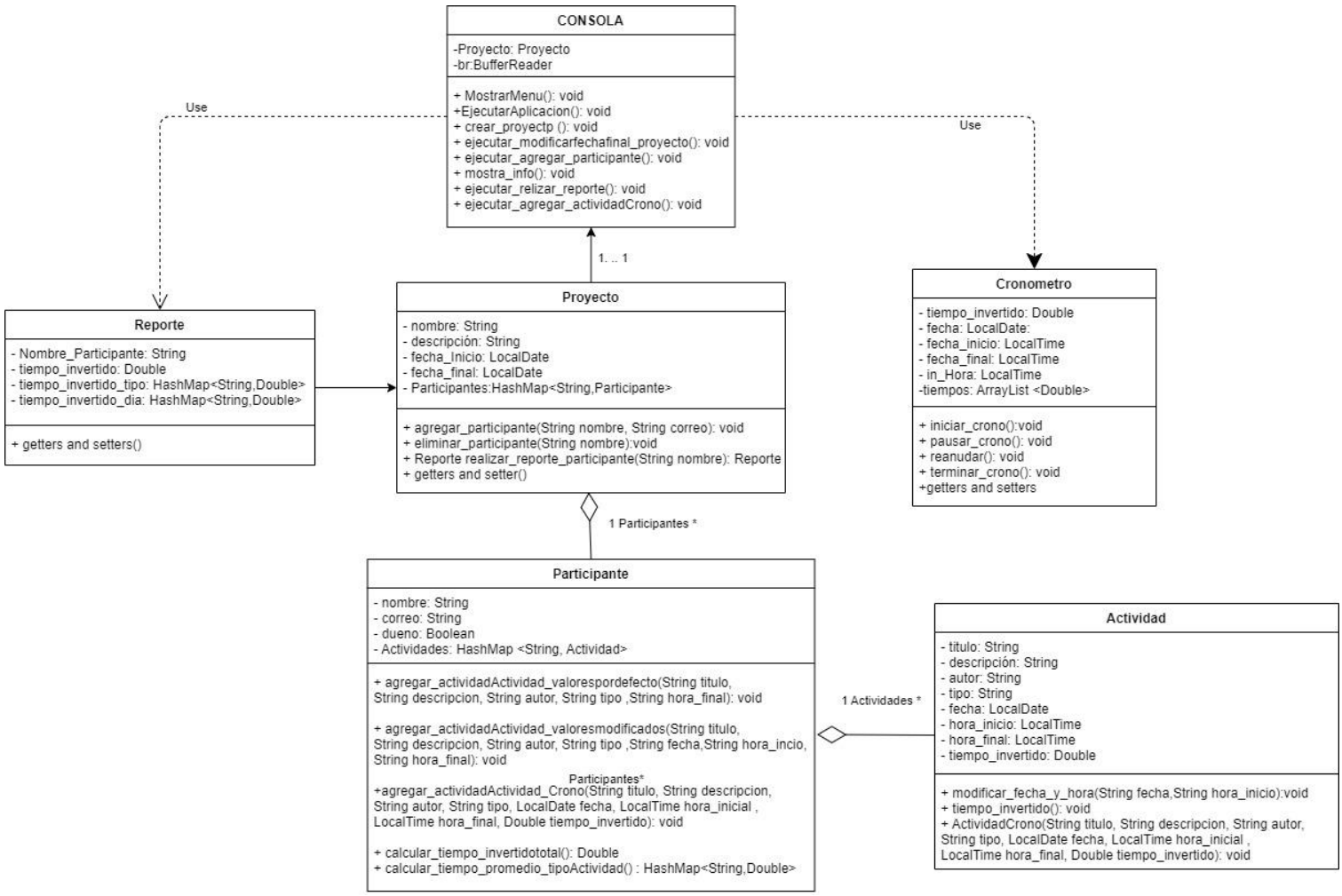
COLABORACIONES

#Responsabilidad => Roles []

1. => [1,6]

2. => [1,6]
3. => [1,2]
4. => [1,2]
5. => [1,6,3,4]
6. => [1,2,4]
7. => [1,2,3]
8. => [1,2,5]
9. => [8,4]

Diagrama UML #4 Iteración (FINAL)



ESTEREOTIPOS:

- **Proyecto:** Structurer, pues es responsable de mantener la estructura, información y ciertos comandos que se deben hacer
- **Participante:** Controller, ya que es el principal responsable del proyecto, así como, el que se encarga de las tareas como hacer commits o generar un reporte en base a una actividad.
- **Reporte:** Information holder, tiene que almacenar los reportes.
- **Actividad:** Structurer, mantiene información sobre las actividades, commits y genera los reportes.

- **Consola:** Interfacer, permite al usuario introducir información y recibir resultados.
- **Cronometro:** Service provider, realiza el cálculo del tiempo de una actividad, almacena y despliega esta información.

Justificación 4 y ultima Iteración

En esta última iteración encontramos grandes dilemas que surgieron en el momento de hacer la implementación y llevaron a tomar ciertas decisiones que cambiaron el modelo de la iteración anterior. Inicialmente El tipo de datos que almacenan horas y fechas se trabajarán de la mano con el módulo de java.time para poder operar tiempos y mantener el formato deseado. Más adelante Se considera que la clase registro no es estrictamente necesaria ya que al analizar los requisitos de la lógica se encontró que dos actividades con el mismo nombre significan que trabajan la misma tarea. En otro momento la relación entre el cronómetro con el resto de los componentes hay que cambiarla ya que su relación no es con el registro si no con la consola ya que es usuario es quien lo operar el cronometro para usar la información obtenida con el fin de hacer un registro de la actividad cronometrada. Y por último se decidió cambiar de ArrayList a HashMaps el almacenamiento de participantes y actividades para ser más efectivos la consulta de información por participante para realizar los reportes. Un último detalle es que las parejas de llave del HashMap de las Actividades HashMap<String,ArrayList<Actividad>> con el fin de tener una lista con todas las actividades que se llamen igual para identificar aquellas de una misma tarea.

Reflexión (Ventajas, Desventajas, Trade-offs):

Con este proceso de diseño se puede decir que logramos un modelo de la lógica con un bajo acoplamiento, los roles están independientes y la lógica está separada de la interacción con los usuarios. Por otro lado, se concluye que la complejidad es bastante baja ya que tiene relaciones organizadas y estructuradas evitando ponerle dificultad en procesos de desarrollo posteriores.

Claramente este modelo no es perfecto, a la hora de implementar será más fácil identificar desventajas donde mediante iteraciones se buscará optimizar aquellas desventajas que pueden estar ligada a relaciones, instanciaciones de clases y demás.

El mejor trade-off es poder separar la interacción con el Usuario de la lógica del dominio, ya que nos permite tener extensibilidad en un futuro para poder actualizar la Interfaz sin necesidad de modificar la lógica y viceversa. Por otro lado, se considera que el hecho de generar el reporte a través de la información administrada por el Proyecto e independientemente de los miembros, permite que se pueda modificar ese reporte mediante un algoritmo.