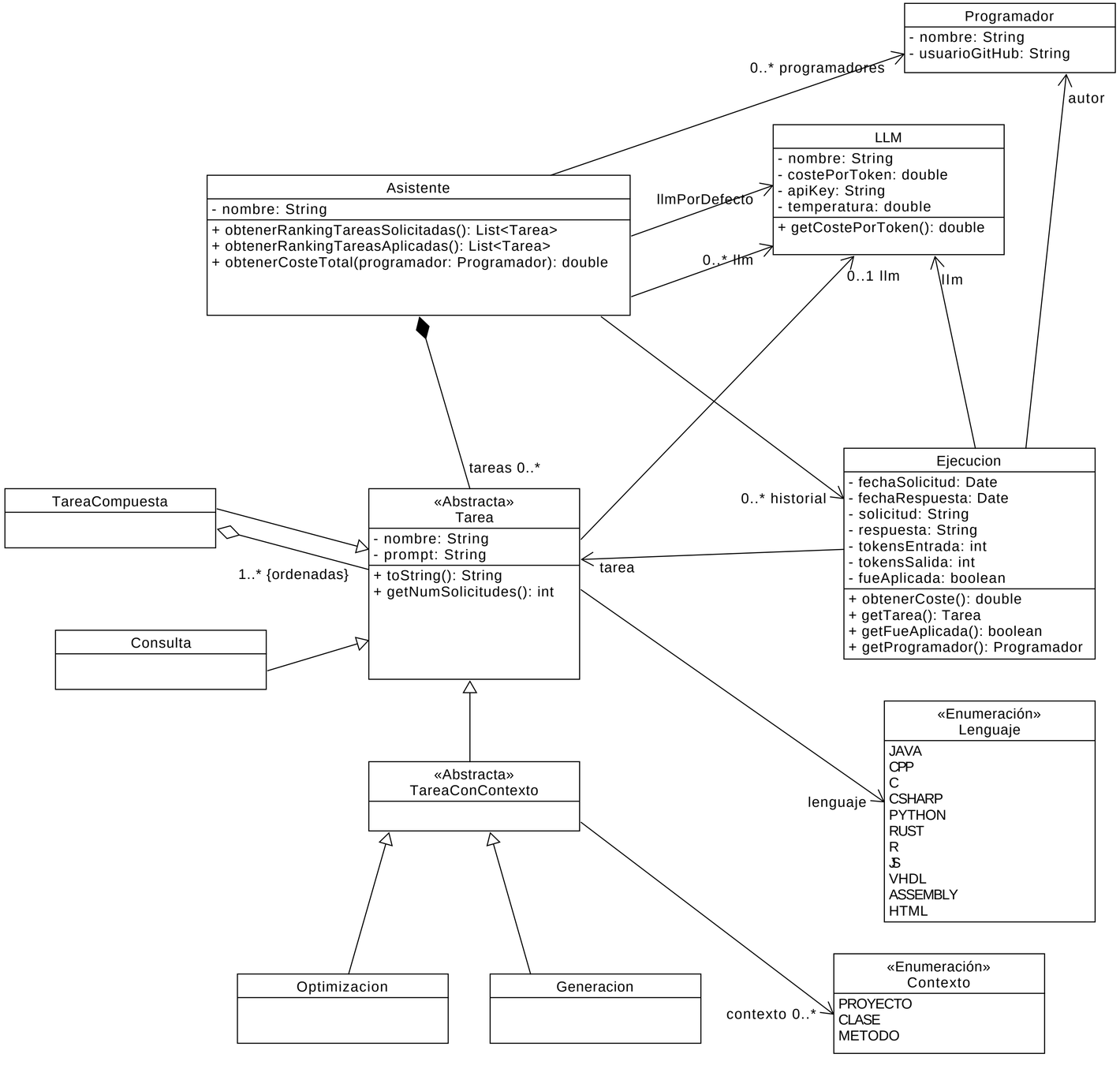

Diseño

Para resolver las tareas compuestas, he planteado una estructura donde una TareaCompuesta tiene una lista ordenada de otras tareas (una agregación). Así resulta fácil anidar tareas unas dentro de otras para que se ejecuten secuencialmente. Además, para no repetir código, he agrupado las tareas de generación y optimización en una clase abstracta TareaConContexto, usando el enumerado Contexto para definir si necesitan leer todo el proyecto, la clase o solo el método actual. Esto asegura extensibilidad en caso de querer modificar las opciones de contexto en un futuro.

Para llevar el registro del historial, he creado la clase Ejecucion. Esta clase guarda quién hizo la petición, los tokens gastados y si la respuesta se aplicó o se descartó. El detalle clave de este diseño es que cada ejecución guarda una conexión directa con el LLM concreto que se usó ese día. De esta forma, si el asistente cambia su modelo por defecto mañana, el cálculo del coste de las ejecuciones pasadas no se rompe ni se recalcula con el precio nuevo.

Por último, haciendo caso a la norma de no llenar el diagrama de getters y setters innecesarios , solo he añadido los estrictamente obligatorios como métodos auxiliares para poder resolver el apartado 2b. Por ejemplo, he expuesto el coste por token en el LLM para calcular el precio de la ejecución, y he añadido métodos en el Asistente para que pueda iterar el historial y sacar los rankings de tareas y el coste por programador.

Métodos

Para implementar el método obtenerCoste() de la clase Ejecución solo se necesita añadir un método auxiliar, ya que el resto de información necesaria se encuentra en la propia clase, lo cual es de hecho el objetivo en diseños orientados a objetos, asegurar que el método se encuentra en la clase con toda la información necesaria.

Para calcular el coste de una ejecución solo se necesita multiplicar el coste/token del llm elegido por el número de tokens usados en la ejecución, es decir, la suma de los tokens de entrada y los de salida.

Método (auxiliar) getCostePorToken en LLM:

```
public double getCostePorToken() {
        return this.costePorToken;
 }
```

Método obtenerCoste() en Ejecucion

```
public double obtenerCoste() {
        int tokensTotales = this.tokensEntrada + this.tokensSalida;
        return tokensTotales * this.llm.getCostePorToken();
 }
```