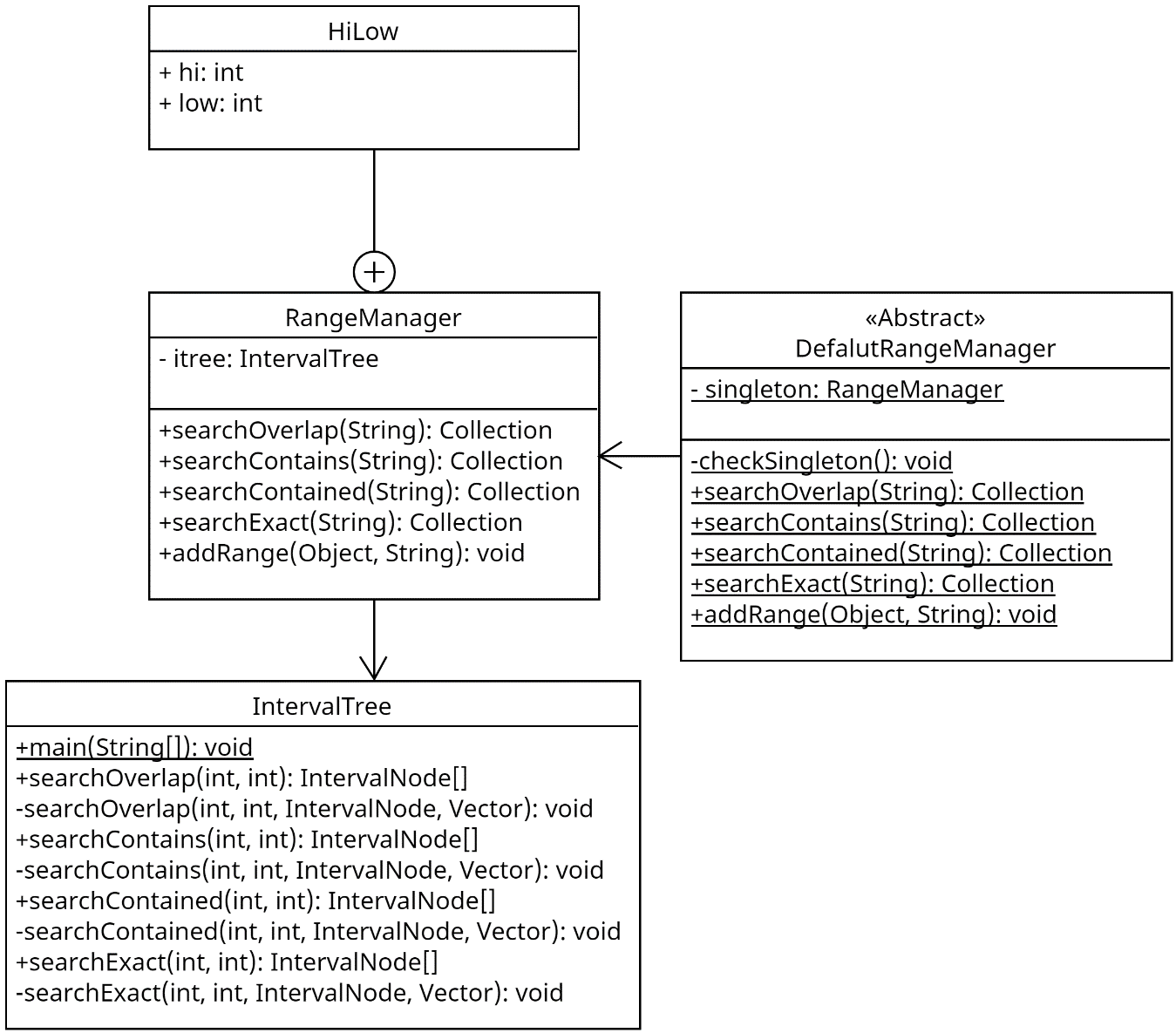
Taller 5 – Patrones

Pablo Peñaranda 201922871

Santiago Gómez 202120215

El proyecto llamado Tsmells desarrollado por Maira B. Carvalho, es una herramienta para la detección automatizada de “olores” de código xUnit de análisis estático, es decir que las pruebas que se llevan a cabo no ejecutan el código que se está analizando. Este proyecto involucra tres pasos: Extraer facts en un modelo RSF, detectar “olores” de código de prueba con crocopat, transformar en GDF y visualizar con Guess. El propósito del código es detectar malas prácticas de programación que puede tener código xUnit, las cuales afectan su sostenibilidad y rentabilidad. Está construido sobre un framework y lenguaje independiente no dependiente del modelo (llamado cdif).

Repositorio: <https://github.com/carvalhomb/tsmells/tree/073753e6ccc710aab0523a37d6d6ca98dae44741>



El diagrama anterior relaciona las clases de que utilizan el singleton para las instancias de los rangos de los árboles utilizados en el proyecto, donde la clase singleton es DefaultRangeManager. La clase singleton (DefaultRangeManager) cuenta con atributos y métodos static. La clase RangeManager contiene otra clase la cual es HiLow y casi los mismos atributos del singleton pero no son static y como atributo crea un IntervalTree que es la otra clase.

Fragmento: <https://github.com/carvalhomb/tsmells/blob/073753e6ccc710aab0523a37d6d6ca98dae44741/guess/guess-src/com/hp/hpl/guess/util/intervals/DefaultRangeManager.java>

El patrón que se analizó en este caso fue el "Singleton". Es un patrón bastante simple utilizado normalmente para asegurarse de que una clase tiene un número especifico de instancias, usualmente una. La idea es que el constructor de la clase sea privado, con el objetivo de que no se puedan crear más instancias de una clase. Además, en este patrón existirá un método que será el encargado de verificar si ya existen la cantidad de objetos que deberían, de no ser así, se crearán automáticamente. Por último, esta clase brinda un punto de acceso global para la (o las) instancias que se creen.

Se suele utilizar este patrón en momentos en los que solo queremos una instancia de una clase específica (ya sea tener solo un administrador en un programa, por ejemplo) o para ahorrar memoria al crea un número único de instancias de un objeto.

Específicamente, dentro del proyecto se utiliza con el propósito de tener una única instancia por defecto de la clase Range Manager. Esto implica que existirá un único rango utilizado para las operaciones de búsqueda sobre los nodos y los árboles que se tienen en esta parte del proyecto. Sin embargo, en este caso en específico existe la posibilidad de modificar el rango con los métodos addRange().

Tiene bastante sentido si lo miramos como una herramienta para ahorrar código o reducir incoherencias, pues será más sencillo tener un rango por defecto que tener que introducir un rango distinto cada vez que queramos realizar las operaciones de búsqueda, más aún si se usarán frecuentemente y no se necesita cambiar el intervalo muy seguido. Esto representa una ventaja en términos de tiempo durante la ejecución de las tareas de búsqueda.

Se podría ver como una desventaja si se quisiera utilizar distintos rangos para cada tipo de búsqueda, árbol, o simplemente para búsquedas diferentes. En ese caso sería bastante más complicado porque se tendría que modificar el objeto cada vez que se quisiera hacer una búsqueda, aumentando el tiempo de ejecución de las mismas.

Otra forma de solucionar el problema que resuelve el singleton en este caso es crear un atributo en la clase IntervalTree que sea un Hash map static y contenga los rangos sobre los cuales trabajarían las instancias de los árboles evitando así la necesidad de crear los rangos. Aunque la implementación y función no sean las mismas, cumpliría para el uso que se le da en la clase IntervalTree de tener un rango por defecto para no tener que crear nuevas instancias si se requiere del uso del mismo.