Ingeniería del Software II

Taller #9 – Verificación usando JML

Deadline: Lunes 19 de Junio 23:59 hs
Fecha de Reentrega: Jueves 6 de Julio 23:59 hs

Introducción

El "*Java Modeling Language*" (**JML**) es un lenguaje de anotaciones para Java que permite especificar propiedades de programas en Java. En este taller vamos a verificar programas en Java usando JML. Todas las anotaciones de JML se escriben como comentarios en el código fuente y comienzan de la forma "`//@ `" (notar el espacio luego del arroba). Recuerde que cuenta con las siguiente anotaciones y *keywords* a la hora de especificar propiedades:

Métodos

- `requires`: especifica las condiciones que deben cumplirse para que un método pueda ejecutarse (i.e., pre-condiciones).
- `ensures`: especifica las condiciones que deben cumplirse al finalizar la ejecución de un método (i.e., post-condiciones). Puede usar la *keyword* `\old` para referirse al valor de una variable antes de la ejecución del método. Además, puede usar la *keyword* `\result` para referirse al valor de retorno del método.
- `assert`: especifica una condición que debe cumplirse en un punto determinado de la ejecución de un método.
- Puede utilizar la anotación `show var1,var2,etc` en algún punto determinado del método para que se muestren los valores que pueden tomar las variables en caso de que ocurra un error de verificación. Es decir, nos provee contraejemplos cuando la verificación falla. Esto es sumamente útil para "*debuguear*".

Loops

- `loop_invariant` (también llamado `maintaining`): especifica las condiciones que deben cumplirse en cada iteración de un ciclo.
- `decreases`: especifica una función de decremento que debe ser decreciente en cada iteración de un ciclo. Se utiliza para demostrar la terminación de un ciclo.
- `loop_writes`: especifica las variables que pueden ser modificadas en cada iteración de un ciclo.

Clases y objetos

- `spec_public`: especifica que un atributo es público para fines de especificación.
- `public invariant`: especifica las condiciones que deben cumplirse para que una instancia de la clase se encuentre en un estado consistente.
- Puede agregar la anotación `pure` a un método de una clase para poder utilizarlo en la especificación de otros métodos. El método marcado como `pure` no debe modificar el estado de la clase.
- Puede acceder a un campo `field` de un objeto `o` con la expresión `o.field`.

Miscelanos

- Puede utilizar las constantes `Integer.MAX_VALUE` e `Integer.MIN_VALUE` para referirse al máximo y mínimo valor de un entero en Java, respectivamente.
- Puede utilizar *keywords* de lógica de primer orden como: `\forall D; R; V;` y `\exists D; R; V;`, donde `D` declara la variable que se va a utilizar en el cuantificador, `R` es un rango para dicha variable y `V` es el predicado booleano que debe ser verdadero.
- Puede utilizar *keywords* de lógica proposicional y aritmética como: `||` (or), `&&` (and), `==>` (implica), `<==>` (sí y sólo sí), `==` (igual a), `!=` (distinto de), `<=` (menor o igual que), `>=` (mayor o igual que), etc.
- Puede utilizar `arr[*]` para referirse a todos los elementos de un arreglo.

Documentación

Puede encontrar más información sobre JML en los siguientes enlaces:

- Página oficial: `https://www.openjml.org/`
- Página de tutoriales con explicación de como usar muchas anotaciones y *keywords*: `https://www.openjml.org/tutorial/`.

Taller

Para este taller, deberá completar varias implementaciones y especificaciones de un proyecto que deberán bajar del campus de la materias.

Antes de comenzar, debe ejecutar el script `setup.sh` que se encuentra en la raíz del proyecto. Este script se encarga de descargar las dependencias necesarias para ejecutar JML. Puede ejecutar el test `testSetup` en el proyecto para verificar que todo esté correctamente instalado. El script está preparado para sistemas operativos Ubuntu versión 18.04 o superior. Puede encontrar otras versiones de JML en `https://github.com/OpenJML/OpenJML/releases`.

Luego, se pide para el taller:

- Completar la especificación de la clase `Absoluto`.
- Completar la especificación de la clase `BusquedaLineal`.
- Completar la especificación de la clase `MapSumaUno`.
- Completar la implentación y especificación de la clase `StackAr`.

Para correr los tests del proyecto puede hacerlo desde una IDE como *IntelliJ IDEA* o desde la terminal con el comando `./gradlew test`.

Formato de Entrega

El taller debe ser subido al campus. Debe ser un archivo zip con el siguiente contenido.

1. Una carpeta con las especificaciones e implementaciones completas y funcionando. Esta implementación debe pasar los tests del archivo `TallerTest.java`. **NO** suba la carpeta `openjml` que se genera al hacer el setup del proyecto.