Tarea 2

Property & Mutation Testing

Parte 1

Objetivos

Después de realizar esta tarea, los alumnos serán capaces de:

En el diseño e implementación de PT:

- **Identificar propiedades** del modelo subyacente al software
    1. Que Unit Testing no es capaz de cubrir adecuadamente.
    2. Describirlas formalmente.
- Implementar una suite de **property tests,** incluyendo:
    1. Generadores de casos de prueba.
    2. Tests de propiedades del modelo.
- **Encontrar fallas** en la implementación a través de testing de propiedades.

En el análisis de PT:

- Entender las ventajas de PT
    - PT como una **generalización** de Unit Testing.
    - ¿Cuál es la **garantía** que me da un PT sobre un componente de software?
- Entender las limitaciones de PT
    - ¿Qué es **intesteable** a través de PT?
    - ¿Qué **rol** cumplen aquí los Unit Tests convencionales?

- Conocer la expresividad de Property Testing

    1. Bajo el contexto de Lógica de Primer Orden
    2. ¿Dónde cae Unit Testing en ese contexto?

A través del uso de una herramienta de Mutation Testing:

- ¿Cómo se testea a un test?

    - ¿Cómo se garantiza la calidad de un test?
    - ¿Cómo se puede medir?

- Entender el rol de MT

    - ¿Qué es lo que revela?
    - ¿Es posible que una test suite elimine al 100% de los mutantes?
    - ¿Qué nos dice eso sobre la expresividad de nuestros tests?

Problema

En esta tarea los alumnos deberán crear una librería vectorial, en Java y en 2 niveles sucesivos de abstracción: *## Qué deben hacer los alumnosFracciones* y luego *Vectores* .

Para asegurarse del correcto funcionamiento de su librería, deberán usar *Property Testing* además de *Unit Testing* . Esto, con el fin de buscar asegurar un comportamiento correcto dado el universo infinito de inputs de su librería.

Para el *Property Testing* de su tarea, usarán *JUnit-QuickCheck* , un módulo para *JUnit* .

Finalmente, en la segunda parte de esta tarea, testearán sus tests usando *Mutation Testing* y evaluarán la calidad de los mismos.

Tarea

Qué recibirán los alumnos

Los alumnos recibirán un proyecto de Java hecho en Eclipse, que ya incluye todas las herramientas de testing necesarias para realizar la tarea.

Además de esto, vienen las clases de la librería vectorial incompletas. Contienen las firmas de los métodos que ustedes deberán implementar y testear.

El programa viene sin tests de la librería. Vienen tests y generadores de ejemplo, para que puedan usar como base para vuestra implementación.

Qué deben hacer los alumnos

En esta primera parte de la Tarea 2, deben concentrarse en implementar la librería y sus tests.

La segunda parte, sobre *Mutation Testing* , comenzará luego de la entrega de la parte 1.

Librería

Los alumnos deberán implementar las clases y sus funciones. En particular:

- `Utils.greatestCommonDenominator` : función del máximo común denominador, importante para simplificar las fracciones.
- `Fraction.normalized` : entrega la fracción de input, simplificada.
- `Fraction.add` : suma dos fracciones.
- `Fraction.sub` : resta una fracción a la otra
- `Fraction.mul` : multiplica dos fracciones
- `Fraction.div` : divide una fracción por la otra
- `Fraction.neg` : entrega el inverso aditivo de la fracción de input
- `Fraction.invMultiplicative` : entrega el inverso multiplicativo de la fracción de input
- `Fraction.one` : entrega la constante equivalente a 1
- `Fraction.zero` : entrega la constante equivalente a 0
- `Fraction.equals` : `True` ssi las 2 fracciones son equivalentes
- `Fraction.gThan` : `True` ssi $a > b$
- `Fraction.lThan` : `True` ssi $a < b$
- `Vector.add` : suma dos vectores
- `Vector.sub` : resta un vector a otro
- `Vector.mul` : multiplica dos vectores componente a componente
- `Vector.div` : divide dos vectores componente a componente
- `Vector.mulScalar` : multiplica un vector por un escalar
- `Vector.divScalar` : divide un vector por un escalar
- `Vector.dot` : producto punto de dos vectores
- `Vector.cross` : producto cruz de dos vectores
- `Vector.zero` : entrega la constante equivalente al Vector Cero
- `Vector.eX` : entrega el vector `eX` : su valor en la primera coordenada es 1 y en el resto es 0.
- `Vector.eY` : entrega a `eY` , equivalente a `eX` pero para la segunda coordenada
- `Vector.eZ` : entrega a `eZ` , equivalente a `eX` pero para la tercera coordenada

- `Vector.equals: True` ssi los dos vectores son equivalentes.

Property Testing

En esta tarea, deberán usar Property Testing para asegurar el correcto funcionamiento de su librería.

Podéis leer sobre Property Testing aquí (https://fr.slideshare.net/ScottWlaschin/an-introduction-to-property-based-testing) y sobre `JUnit-Quickcheck` aquí (https://examples.javacodegeeks.com/core-java/junit/junit-quickcheck-example/)

Para hacer uso de Property Testing, necesitarán de dos cosas: **Generadores** y **Propiedades**.

Generadores

Los Generadores son esenciales para hacer Property Testing. Un Generador **genera input**, que es luego **testeado** contra una **propiedad**.

Los Generadores son una representación del dominio de input que quieren testear.

Viene un generador de ejemplo en el proyecto:
`src/test/java/tarea02/NaturalGenerator.java`

A la hora de crear un generador de elementos del tipo `T` hay 3 pasos:

- Clase: `public class TGenerator extends Generator<T>.`
- Constructor: `public TGenerator() { super(T.class); }`
- Método generador:

```
public T generate(SourceOfRandomness source, GenerationStatus gStatus
        return new T(
                    // Parameters of T's constructor go here
                );
    }
```

Viendo el ejemplo incluido en el proyecto pueden entender cómo usar `source` para generar elementos de las clases que programen.

Propiedades

Sus tests ejecutarán `asserts` sobre elementos que salgan de sus Generadores. En particular, esos asserts serán propiedades que los elementos generados deben cumplir.

Si falla un assert, la librería os entregará el elemento que no pudo cumplir la propiedad.

Veamos un ejemplo. Al igual que en las diapositivas del primer enlace, la suma de fracciones es conmutativa. Esta propiedad tiene mucho peso, y pueden aprovecharla de la

siguiente forma:

- En lugar de probar, por ejemplo: `assertEquals(1 + 2, 2 + 1)`,
- Usando el generador de fracciones, ejecutan el test: `conmutatividad(a, b) := assertEquals(a + b, b + a)`

Pueden ver una implementación de distintos tests de propiedades, en `src/test/java/tarea02/NaturalProperties.java`, que prueba propiedades de los números naturales. Éstos se encuentran implementados en `src/main/java/tarea02/Natural.java`. Sigan los comentarios de los tests para entender los detalles de implementación.

Una vez que empiecen a trabajar con sus propiedades, para cada Propiedad que usen, deberán documentarla:

1. Definición de la propiedad.

   $Conmutatividad := a + b = b + a$

2. Justificación: ¿por qué esta propiedad complementa a las otras propiedades que ya testean?. Por ejemplo, quizás si ya testearon que:

   - $sucesor(a) > a$ y que
   - $antecesor(sucesor(a)) = a$,
   - No debiera ser necesario testear que $antecesor(a) < a$

Paso a paso

Recomendamos seguir esta lista de pasos para cumplir con vuestro objetivo:

- Estudiar las propiedades de los números Racionales, y entender qué propiedades caracterizan a sus operaciones.
- Formalizar las propiedades usando lógica de primer orden. Por ejemplo, "Para todo x en los Racionales, $x \cdot x \geq 0$". Esto les permitirá aclarar qué es lo que deben implementar en sus tests.
- Programar los números Racionales para familiarizarse con la sintaxis.
- Implementar sus tests, ahora que entienden la sintaxis de la librería.
- Corregir los errores que encuentren.

Recomendamos continuar con los Vectores, una vez que consideren terminada la parte de fracciones de la librería:

- Estudiar las propiedades de los Vectores en 3 dimensiones.
- Al igual que antes, formalizarlas en lógica de primer orden. Ahora que tienen

fracciones y vectores, pueden usar propiedades con ambos. Por ejemplo, "Para todo vector V, dado z = Fraction.zero(), se cumple que $V \cdot z$ = Vector.zero()".
- Implementar sus tests antes de implementar la librería. Los tests fallarán, y eso es normal.
- Programar los Vectores hasta que los tests pasen.

Evaluación

Esta tarea será evaluada de la siguiente manera, por el momento:

- Generadores: 1.5 puntos
- Implementación de la Librería: 1.5 puntos
- Testing de propiedades: 1.5 puntos
- Análisis de Property Testing: 2 puntos

Generadores

Obtendrán 0.75 puntos por cada generador (los de `Fraction` y `Vector`). En particular:

- 0.5 por generar elementos correctamente. Por ejemplo, $\frac{1}{0}$ no es una fracción correcta.
- 0.25 por cubrir todo el dominio con su generador. Si por ejemplo su generador no puede crear fracciones negativas, el dominio no es completo.

Librería

Se pondera el puntaje por el porcentaje de correctitud: $pCorrec = \frac{\#métodoscorrectos}{\#métodostotales}$

- 0.75 puntos por implementar `Fraction`
- 0.75 puntos por implementar `Vector`

Testing de propiedades

Se otorgará 1 punto por los Tests de `Fraction` y 1 punto por los Tests de `Vector`. Para ambos, se evaluará que los tests:

- **(0.25)** Estén documentados con claridad
- **(0.25)** Sean adecuados, es decir, que prueben propiedades existentes.
- **(0.25)** Estén correctamente implementados
- **(0.25)** Cubran la funcionalidad de las clases: que no falten propiedades esenciales.

Análisis

Al momento de entregar su tarea, deberá analizar Property Testing, tratando de responder a su elección algunas de las siguientes preguntas:

- **1.A** ¿Cuáles son sus ventajas?
    - **1.B** ¿Qué garantías entrega Property Testing sobre un componente de software?
- **2.A** ¿Cuáles son sus limitaciones?
    - **2.B** ¿Son necesarios los Unit Test si se tiene acceso a Property Tests?
- **3.A** Justifique la idea de *"representar las propiedades testeadas a través de lógica de primer orden"*
    - **3.B** ¿Qué forma de lógica corresponde a Unit Testing bajo este criterio?

El alumno puede elegir entre:

1. Responder 2 preguntas de tipo **A** , con 2 ejemplos cada una, o
2. Responder 2 preguntas del mismo número (**A** y **B**), con 2 ejemplos cada una.

Ambas formas de responder le otorgan 2 puntos al alumno.

Nota

La nota se calculará usando la siguiente fórmula:

$Nota = 1 + puntos\_totales$

Es decir:

$Nota \leq 7.5$

La nota máxima es a propósito mayor a 7. Aprovechen cada punto de esta tarea, tanto por su nota como para aprender de Testing ;)

Esta tarea admite Bonus. Revise el anexo al final de este enunciado para más detalles.

Entrega

Tiempos

Los alumnos tendrán 2 semanas desde el primer martes luego de subirse este enunciado. Es decir, el plazo comienza el 10 de Octubre y termina el 24 de Octubre.

Cómo entregar

Al igual que en la tarea 1, vuestras respuestas de análisis es preferible que las dejen en un archivo de nombre `README` . La extensión y formato *(pdf, txt, md)* no son importantes. Usen lo que les acomode para escribirlas.

En una carpeta comprimida dejen sus respuestas junto con la carpeta de su proyecto.

Para entregarla, se habilitará un buzón de tareas en el Siding. El buzón admite entregas múltiples por si prefieren entregar versiones preliminares de su tarea mientras terminan lo que les falte :)

Soporte

Usen el Slack del curso para sus dudas. Mi mail si no pueden *(ffischer@uc.cl)* , pero tengan en cuenta que dedicaré mi tiempo principalmente al Slack, pues así puedo ayudarlos a todos con mayor facilidad.

Bibliografía

- An introduction to Property-based Testing (https://fr.slideshare.net/ScottWlaschin/an-introduction-to-property-based-testing)
- JUnit-Quickcheck example (https://examples.javacodegeeks.com/core-java/junit/junit-quickcheck-example/)

Anexo: Bonus

Esta sección no está terminada. Será actualizada después de terminar la sección de evaluación