

Ingeniería del Software II

Taller #2 – Random Testing

LEER EL ENUNCIADO COMPLETO ANTES DE ARRANCAR.

Fecha de entrega: 11 de Abril de 2024

Fecha de re-entrega: 2 de Mayo de 2024 (no hay extensiones)

Herramienta Randoop

Randoop es una herramienta de generación automática de casos de test unitarios para Java (en formato JUnit). Esta herramienta implementa una técnica de generación de casos de tests aleatorios, guiados por retroalimentación (*feedback-directed random testing*¹). Cada caso de test consiste en una secuencia de llamadas a métodos de la clase bajo prueba, seguida de una aserción que captura el comportamiento esperado de la clase bajo prueba. Randoop puede ser utilizado con dos propósitos: encontrar errores en un programa y crear tests de regresión para detectar si cambia el comportamiento de su programa en el futuro.

En este taller, utilizaremos Randoop para generar automáticamente casos de test unitarios para la clase **StackAr** (vista en el taller anterior). Los siguientes links proveen más información al respecto de esta herramienta:

- Website de Randoop: <https://randoop.github.io/randoop/>
- Manual de uso de Randoop: <https://randoop.github.io/randoop/manual/index.html>

Herramienta Pitest

Pitest es una herramienta para generar mutantes de un programa Java y calcular el **mutation score** de un *test suite*. En este taller, utilizaremos Pitest para generar automáticamente mutantes de la clase **StackAr** y calcular el **mutation score** de un *test suite* generado por Randoop. Los siguientes links proveen más información al respecto de esta herramienta:

- Operadores de mutación disponibles en Pitest: <https://pitest.org/quickstart/mutators/>
- Cómo utilizar Pitest por línea de comandos: <https://pitest.org/quickstart/commandline/>
- Cómo utilizar Pitest con el *plugin* de Gradle: <https://gradle-pitest-plugin.solidsoft.info/>

Setup y contenido del taller

Descargar el proyecto *StackAr* del campus, e importarlo en la IDE IntelliJ IDEA. Este proyecto contiene una implementación de un *stack* (pila) en Java.

Para generar los tests unitarios con Randoop, ejecutar la *Gradle task* **randoop**. Esto puede hacerlo desde la IDE abriendo el panel *Gradle* y haciendo doble-click sobre la tarea **randoop** (en la sección de *verification*). Los archivos generados por Randoop se encuentran en la carpeta **src/test/java** del proyecto, con el nombre **RegressionTest#.java**. Tenga en cuenta que cada archivo puede contener más de un test. Para inspeccionar la configuración de la herramienta Randoop, puede revisar los argumentos de la *Gradle task* **randoop** en el archivo **build.gradle**.

Una vez generados, puede correr los tests unitarios desde la IDE haciendo click derecho sobre el módulo test y seleccionando *Run Tests*. El reporte de *coverage* generado por JaCoCo al finalizar los tests se encuentra en el archivo **build/reports/jacoco/test/html/index.html**.

¹Feedback-directed random test generation by Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. In ICSE '07: Proceedings of the 29th International Conference on Software Engineering.

Para correr **Pitest**, ejecutar la *Gradle task* **pitest**. Esto puede hacerlo desde la IDE abriendo el panel *Gradle* y haciendo doble-click sobre la tarea **pitest** (en la sección de *verification*). El output del report de **Pitest** queda en `build/reports/pitest/index.html`. Para inspeccionar la configuración de la herramienta **Pitest**, puede revisar los argumentos de la *Gradle task* **pitest** en el archivo `build.gradle`.

Dado que **Pitest** trabaja a nivel de bytecode, vamos a utilizar la herramienta **Java Decompiler** para convertir el bytecode generado por **Pitest** a código fuente **Java**, y poder inspeccionar los mutantes generados. Esto ya se encuentra configurado en el archivo `build.gradle`. Luego de correr **Pitest**, puede encontrar los mutantes generados en la carpeta `build/reports/pitest/export/org/autotest/StackAr/mutants`.

Ejercicio 1

Ejecutar **Randoop** sobre la clase **StackAr** para que genere todos los tests aleatorios posibles durante 15 segundos. Luego, responder:

- ¿Cuántos casos de test produjo **Randoop**?
- ¿Hay casos de test que fallan?
- ¿Cuál es el *instruction coverage* alcanzado por los tests generados para la clase **StackAr**?

Ejercicio 2

Completar el método `repoOK()` de la clase **StackAr** para que retorne **true** solamente si la estructura del **StackAr** es válida. Tener en cuenta que el código de este método no debe tirar excepciones. Una instancia de **StackAr** es válida sii:

- $elems \neq null$
- $readIndex \geq -1$ y $readIndex < elems.length$
- $\forall i > readIndex, elems_i = null$

Luego,

- Ejecutar **Randoop** por 1 min sobre **StackAr** y correr los tests generados. ¿Hay casos de test que fallan? ¿Cuántos? Si hay tests que fallan, analizarlos y explicar porqué fallan.
- Si hay casos de tests que fallan, reparar el programa **StackAr** para que pasen los tests. Luego, volver a ejecutar **Randoop** con un minuto de presupuesto y cerciorarse de que no reporte más casos de tests que fallan.
- Reportar el *instruction coverage* alcanzado por los últimos casos de tests generados por **Randoop** para la clase **StackAr**.

Ejercicio 3

Ejecutar **Pitest** sobre el último test suite generado por **Randoop**. Responder:

- ¿Cuántos mutantes construye **Pitest**? ¿Cuál es el *mutation score*?
- Extender manualmente el test suite para obtener el mejor *mutation score* posible con **Pitest**. Ignore los mutantes que pudiera haber generado **Pitest** para el método `repoOK()`. ¿Cuál es el mejor *mutation score* que pudo obtener? Si hubiera mutantes equivalentes, explique cuáles son y justifique porqué son equivalentes.

Formato de Entrega

El taller debe ser entregado en el campus de la materia. La entrega debe incluir un archivo `entrega.zip` con el código modificado del proyecto *StackAr* que descargaron del campus. Este archivo debe incluir:

- El archivo **StackAr** con sus modificaciones correspondientes.
- Los tests generados para el ejercicio #1, en una carpeta `ejercicio1` dentro de `src/test/java/org/autotest`.
- Los últimos tests generados para el ejercicio #2, en una carpeta `ejercicio2` dentro de `src/test/java/org/autotest`.
- Los tests que escribieron para mejorar el **mutation score** en el ejercicio #3, en una carpeta `ejercicio3` dentro de `src/test/java/org/autotest`.
- El reporte de *JaCoCo* para el *test suite* final que hayan armado (el del ejercicio #3).
- El archivo `RESPUESTAS.txt` con las respuestas a las preguntas de los ejercicios.