

Taller #3: Randoop

Generación Automática de Casos de Tests - 2018

Random Testing



- **Idea:** definamos nuestro “mono tipeador” y probemos nuestro programa usando sus inputs
- Necesitamos para ello de un Oráculo

Random Testing para programas Orientados a Objetos

- Dado este método (Java)

```
public static Integer largest(LinkedList<Integer> list) {  
    int index = 0;  
    int max = Integer.MIN_VALUE;  
    while (index <= list.size()-1) {  
        if (list.get(index) > max) {  
            max = list.get(index);  
        }  
        index++;  
    }  
    return max;  
}
```

- La solución anterior puede usar `LinkedList()` o `null`
 - `list=null`
 - `list=[]` (la lista vacía)

Random Testing guiado por Feedback



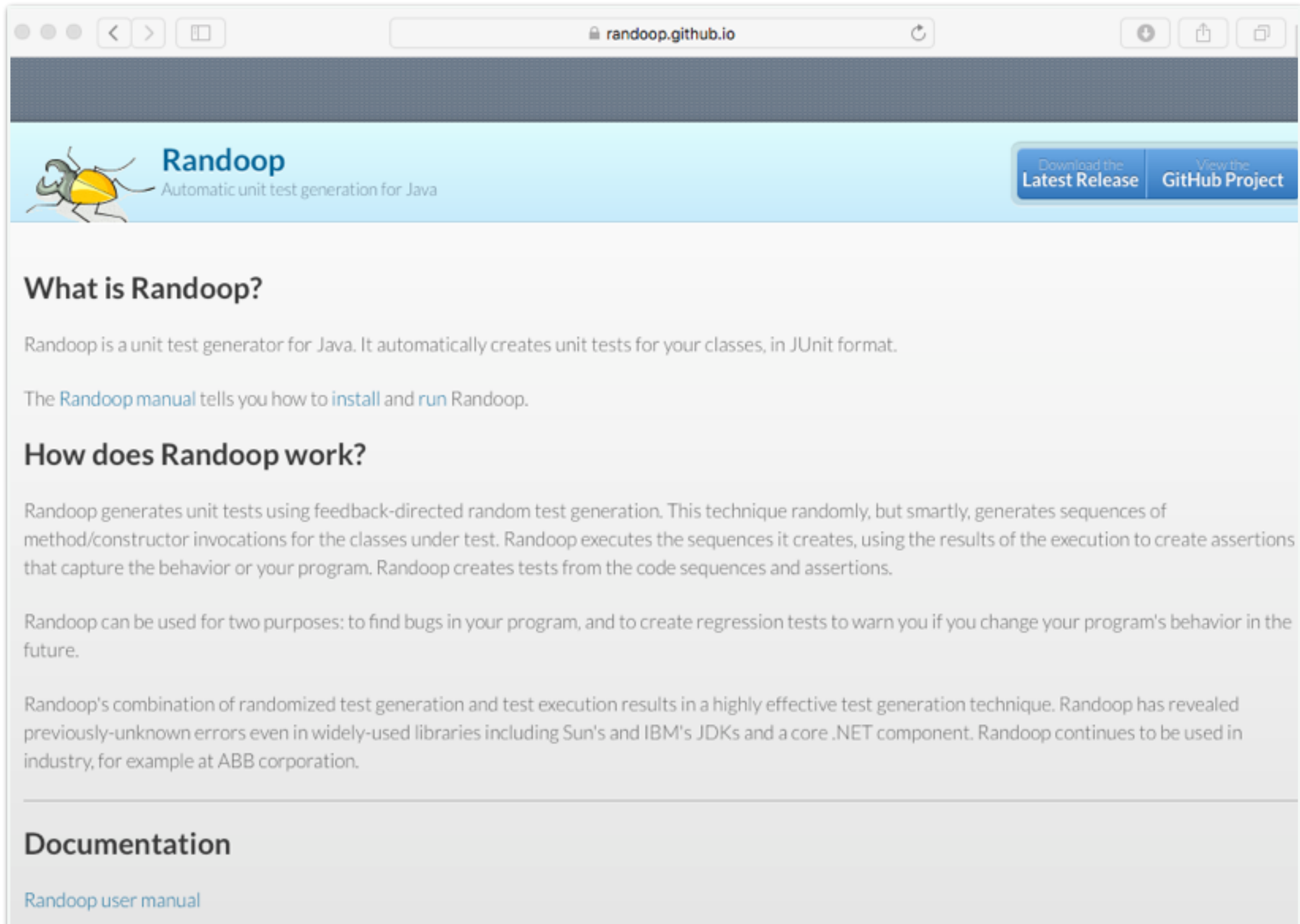
Feedback de la ejecución del Test

```
// Catalog of methods of interest  
C = {...}  
// seed basic sequences (e.g. "boolean b=false", "int i=0")  
seqs = { ... }  
// valid instances  
instances = {}  
// exceptional tests  
exception_seqs = {}
```

```
Do until time limit expires:  
    new_test = generate_new_test_sequence(C,seqs):  
    try:  
        r = execute_test(new_test)  
        if (r !in instances)  
            // sequence creates a previously unseen instance  
            seqs.add(new_test)  
            instances.add(r)  
        else:  
            // discard sequence  
    catch Exception:  
        // sequence signals an exception  
        exception_seqs.add(new_test)  
return seqs, exception_seqs
```

Usamos el feedback de la ejecución

Randoop



The screenshot shows a web browser window with the address bar displaying `randoop.github.io`. The page has a light blue header with the Randoop logo (a cartoon beetle) and the text "Randoop Automatic unit test generation for Java". To the right of the header are two buttons: "Download the Latest Release" and "View the GitHub Project". The main content area has a white background and contains the following sections:

What is Randoop?

Randoop is a unit test generator for Java. It automatically creates unit tests for your classes, in JUnit format.

The [Randoop manual](#) tells you how to [install](#) and [run](#) Randoop.

How does Randoop work?

Randoop generates unit tests using feedback-directed random test generation. This technique randomly, but smartly, generates sequences of method/constructor invocations for the classes under test. Randoop executes the sequences it creates, using the results of the execution to create assertions that capture the behavior of your program. Randoop creates tests from the code sequences and assertions.

Randoop can be used for two purposes: to find bugs in your program, and to create regression tests to warn you if you change your program's behavior in the future.

Randoop's combination of randomized test generation and test execution results in a highly effective test generation technique. Randoop has revealed previously-unknown errors even in widely-used libraries including Sun's and IBM's JDKs and a core .NET component. Randoop continues to be used in industry, for example at ABB corporation.

Documentation

[Randoop user manual](#)

Randooop



- **Random Testing** guiado por feedback para el lenguaje orientado a objetos Java
- Genera automáticamente un conjunto de *test classes* **JUnit** (puede generar muchos tests en muy poco tiempo y exceder el límite de compilación)
- Stándard de-facto de Random Testing para Java

Randooop

- Open-Source (<https://github.com/randooop/randooop>)
- Documentado (<https://randooop.github.io/randooop/manual/>)
- Trabaja sobre el bytecode (no necesita source-code)

Randooop: Valores Primitivos

- byte: -1, 0 1, 10, 100
- short: -1, 0 1, 10, 100
- int: -1, 0 1, 10, 100
- long: -1, 0 1, 10, 100
- float: -1, 0 1, 10, 100
- double: -1, 0 1, 10, 100
- char: '#', ' ', '4', 'a'
- java.lang.String: "", "hi!"

Randooop ofrece distintas formas de agregar nuevas opciones de valores primitivos

Randooop: Generación por Línea de Comando

- `java -ea -classpath randoop-all-4.1.0.jar:[projectCP]`
`randoop.main.Main gentests`
`--testclass=[ClassName]`
`--timelimit=[segundos]`

Randooop: Generación por Línea de Comando

- `--classlist=[file.txt]`
- `--junit-out-dir=[dirName]`
- `--testsperfile=[NroDeTests]` (default 500)
- `--maxsize=[int]` (default 100)
- `--randomseed=[int]` (default 0)

Randoop: Oráculos

- Object.equals():
 - **Reflexividad**: `o.equals(o) == true`
 - **Simetría**: `o1.equals(o2) == o2.equals(o1)`
 - **Transitividad**: Si `o1.equals(o2) && o2.equals(o3)` entonces `o1.equals(o3)`
 - **Nullity**: `o.equals(null) == false`



Randooop: Oráculos

- Object.hashCode():
 - ***Equals y hashCode son consistentes:*** Si
 `o1.equals(o2)==true`, entonces
 `o1.hashCode()==o2.hashCode()`
 - No tira una excepción
- Object.clone():
 - No tira una excepción
- `Object.toString()`:
 - No tira una excepción



Randooop: Oráculos

- [Comparable.compareTo\(\)](#) y [Comparator.compare\(\)](#):
 - **Reflexividad**: `o.compareTo(o) == 0` (implicado por anti-simetría)
 - **Anti-simetría**: `sgn(o1.compareTo(o2)) == -sgn(o2.compareTo(o1))`
 - **Transitividad**: Si `o1.compareTo(o2) > 0` && `o2.compareTo(o3) > 0` entonces `o1.compareTo(o3) > 0`
 - **Sustitución de equals()**: Si `x.compareTo(y) == 0` entonces `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`
 - **Consistencia con equals()**: `(x.compareTo(y) == 0) == x.equals(y)`
 - No tira excepción



Randooop: Oráculos

- Se produjo un *NullPointerException* cuando únicamente se utilizan argumentos que son distintos de null.
- Se produjo un *OutOfMemoryError*
- Se produjo un *AssertionFailure* (si *-ea* está activo)

Todos los oráculos se pueden activar y desactivar antes de la generación

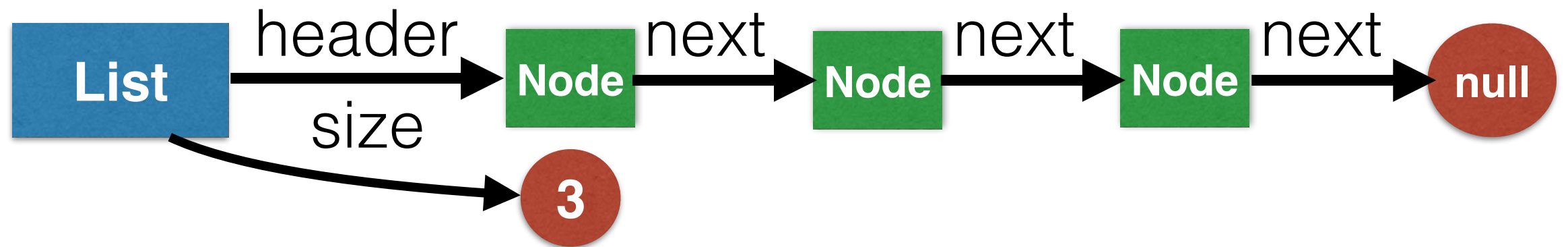
Randooop: Resultado

- Escribe las test classes en la carpeta indicada usando la opción *--junit-output-dir*
- Las test classes llevan los nombres:
 - **Passing Tests**: RegressionTest0.java, RegressionTest1.java, etc...
 - **Failing Tests**: ErrorTest0.java, ErrorTest1.java, etc...

Oráculo @CheckRep

- La anotación @CheckRep permite indicarle a Randoop que ese método debe ser chequeado antes y después de cada invocación a esa clase
- El método booleano debe:
 - No tener efectos colaterales
 - Ser público, no estático y sin parámetros
 - Retorna false si el invariante de representación de la instancia está roto

Ejemplo: CheckRep



@CheckRep

```
public boolean repOK() {  
    Set<Node> visited = new HashSet<Node>();  
    Node curr = this.header;  
    while (curr!=null) {  
        if (visited.contains(curr))  
            return false;  
        visited.add(curr);  
        curr = curr.next;  
    }  
    return visited.size()==this.size;  
}
```

Randoop



- Randoop ejecuta el código sin ninguna protección
- Puede dar lugar a efectos indeseados si la clase bajo test modifica el file system o altera el sistema de algún modo
- Ejemplo `Foo.deleteAll(String str)`

Descargas

- Descargar la última versión de Randoop (4.1.0) de su página web y copiar el jar-file
- Descargar el archivo en el campus

Carpetas



randoop-all-4.1.0.jar



stackar

pom.xml

src

Preparación



- Moverse a la carpeta “stackar”
- Compilar el código ejecutando `$ mvn clean install`

#1: Ejecutando Randoop

- Generar test cases usando por línea de comando:

```
$ java -ea -classpath ../randoop-all-4.1.0.jar:target/classes  
randoop.main.Main gentests --testclass=org.autotest.StackAr --  
timelimit=15 --testsperfile=500 --junit-output-dir=src/test/java
```

- ¿Cuántos passing test cases produjo Randoop?
- ¿Hay failing tests?



#2: Cobertura

- Ejecutar JaCoCo usando el test suite generado por Randoop
- ¿Cuántas líneas son cubiertas?
- ¿Cuántos branches son cubiertos?



#3: Oráculos en Randoop

- Completar el método `StackAr.repOK()` para que retorne `true` solamente si la estructura del `StackAr` es válida
- Una instancia de `StackAr` es válida sii:
 - `elems!=null`
 - `readIndex >= -1` y `readIndex<elems.length`
 - `elems[i]==null` para todo `i>readIndex`
- Ejecutar Randoop nuevamente por 15sec ¿hay failing tests?
Modificar el programa hasta que no encuentre failing tests