

# Generación Automática de Tests

Ingeniería del Software 2

Juan P. Galeotti



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA



**ICC**

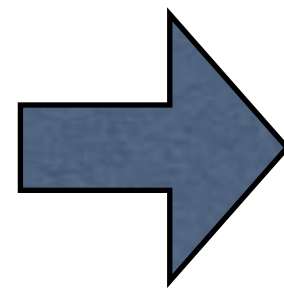
Instituto de Ciencias  
de la Computación

# Repaso: Automatic Test Case Generation

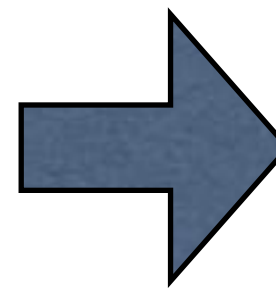
Software



Test Suite



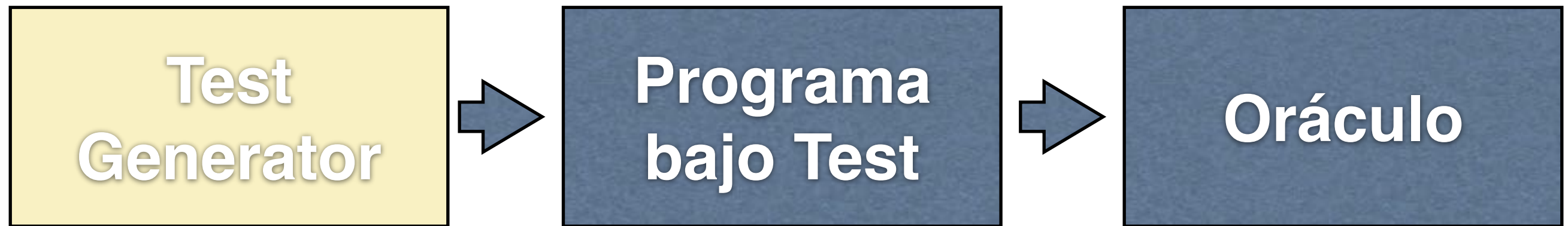
Test  
Generator



```
@Test
public void test0() throws Throwable {
    Foo foo0 = new Foo();
    Bar bar0 = new Bar("baz3");
    bar0.coverMe(foo0);
    assertEquals(0, foo0.getX());
}
```



# Repaso: Escenario #1: Detección de fallas



- **Idea:** para poder clasificar una falla necesitamos un **oráculo**
- ¿Qué pasa si no tenemos un oráculo?

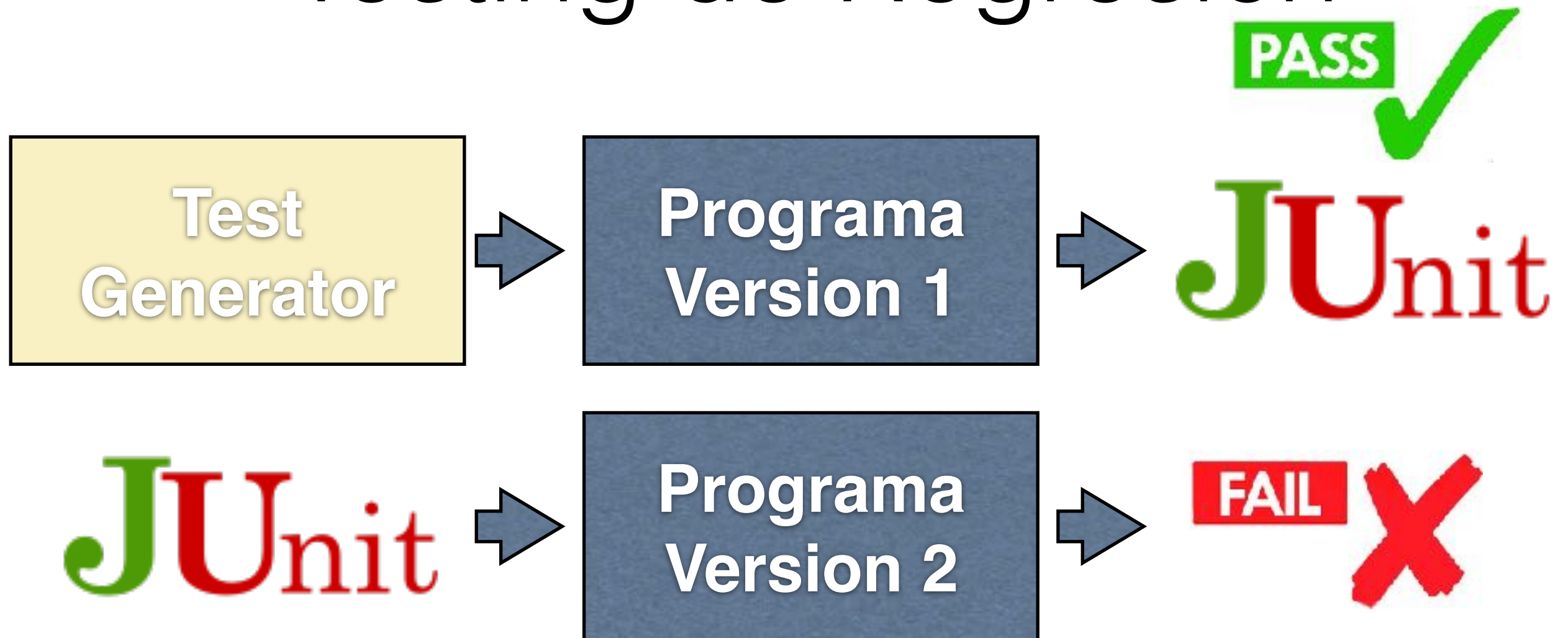
# Repaso: Escenario #2: Testing de Robustez

- A veces no disponemos de un oráculo para chequear automáticamente el programa



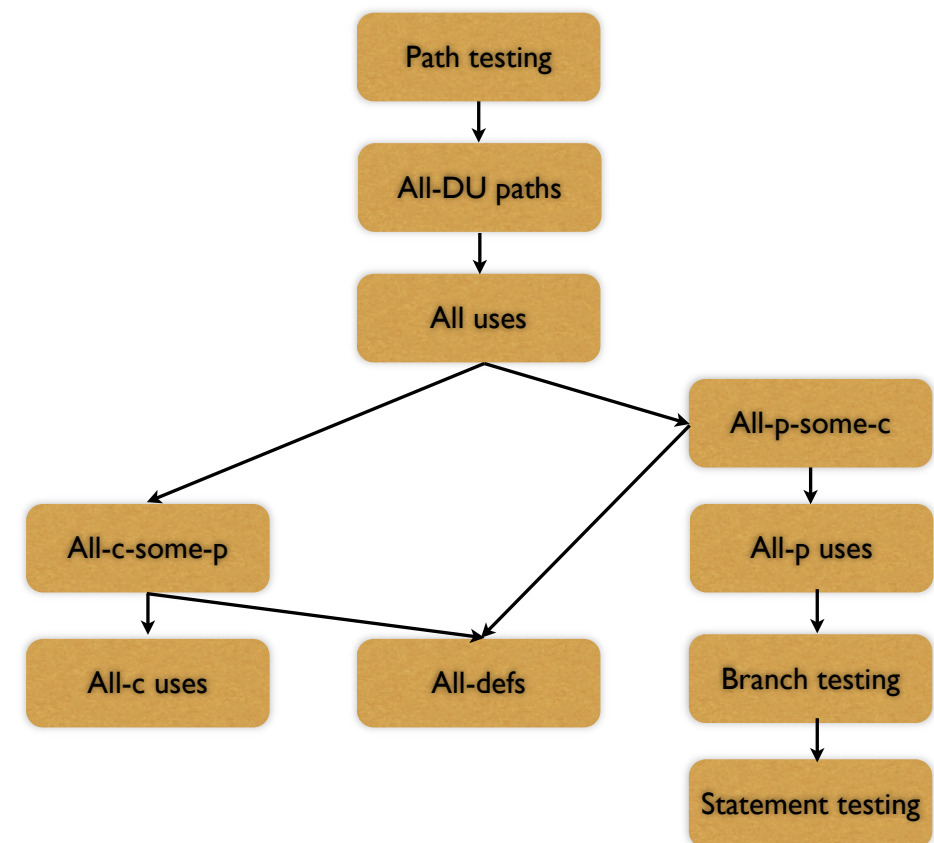
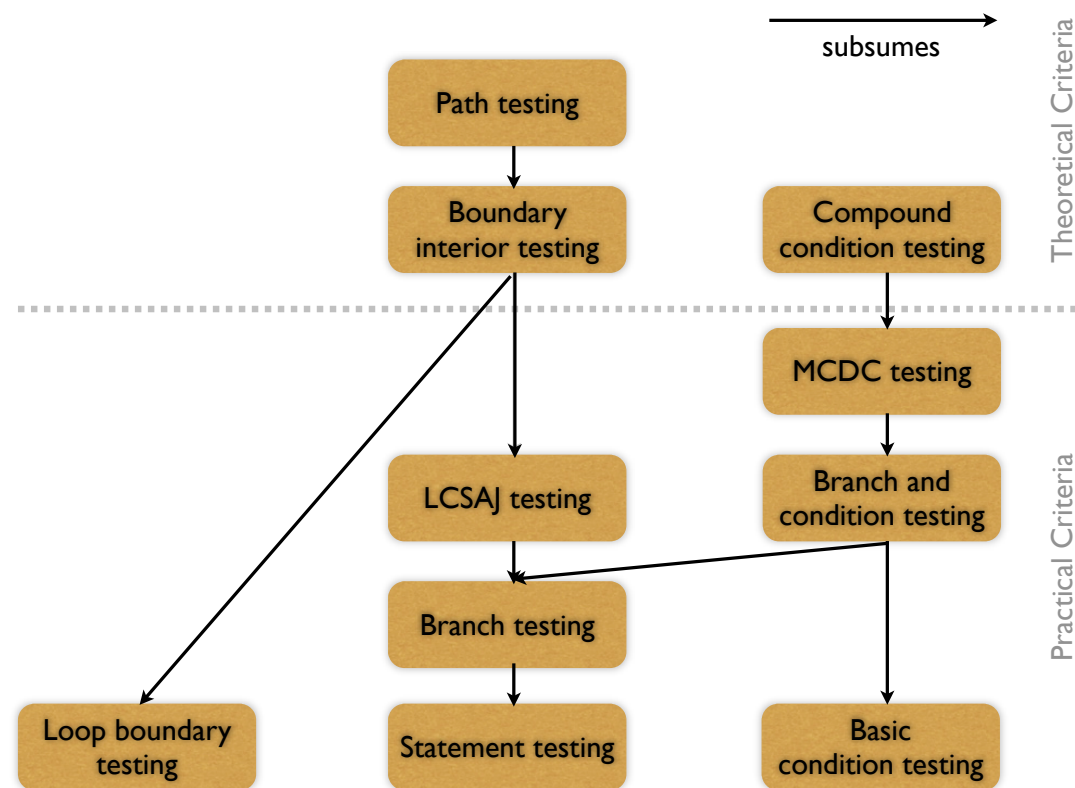
- Robustness Testing: ¿ocurrió algún crash?
- ¿Qué es un crash? Oráculos implícitos

# Repaso: Escenario #3: Testing de Regresión

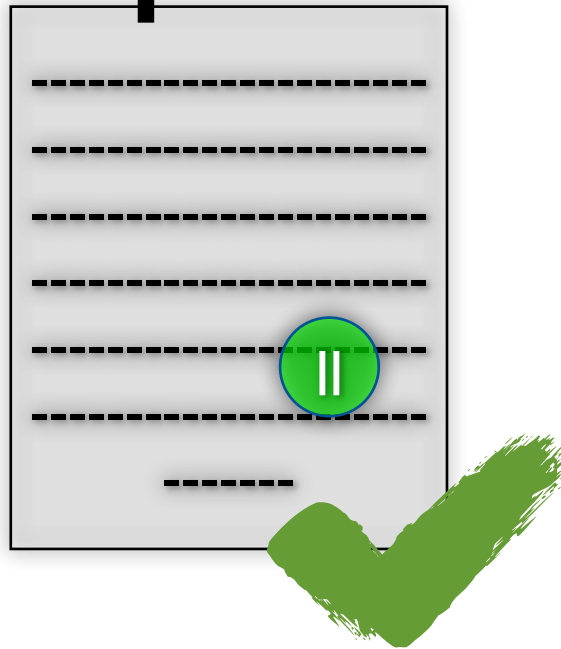


- Regression Testing: ¿hay una diferencia?
- ¿Es una diferencia esperable?

# Repaso: Criterios de Adecuación



# Repaso: Mutation Analysis

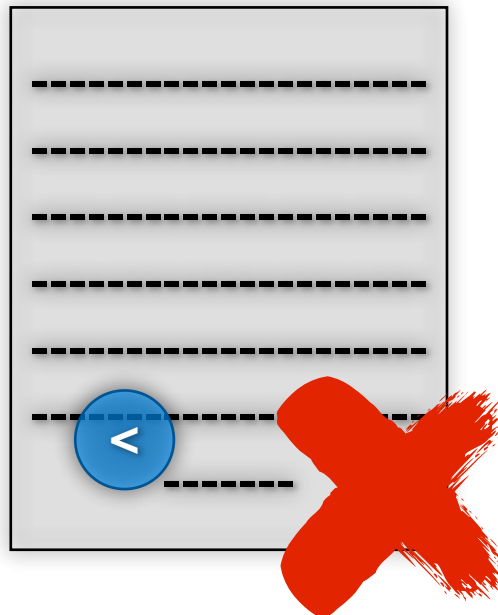


Mutation Score:

Mutantes Muertos

---

Total Mutantes





HASTA ACÁ ES  
REPASO

# Repaso: Mutantes Equivalentes

- Mutación = cambio sintáctico
- El cambio puede dejar la semántica inalterada
- Los Mutantes Equivalentes son difíciles de detectar (problema indecidible)
- Pueden ser alcanzados, pero quizás no se infecte el estado del programa
- Pueden producir una infección, pero sin propagación





# Automatic Generation of Tests

## ↳ Random Testing

- `triangle(int,int,int)` recibe 3 números enteros
- ¿Cómo podemos probar nuestro programa 1000 veces?
- **Random Driver**: ejercita el programa usando valores aleatorios

↳ provide random values to test

static analysis  
of the program

# Random Driver

**Let**  $M(p_0:T_0, \dots, p_k:T_k)$  be a program

*example program.*

**while** *budget is not empty*

*budget to execute test*

**For each** input parameter  $p_i:T_i$

**If**  $T_i$  is primitive  $v_i := \text{get random } T_i$

**Else**  $v_i := \text{null}$

*primitive data type*

**Add**  $M(v_0, \dots, v_k)$  **to** tests

**Return** tests

mientras tiempo presupuesto,  
por cada primitiva genero  
valores random p/ los parametros

# v2 random test generation

```
Let  $M(p_0:T_0, \dots, p_k:T_k)$  be a program  
while budget is not empty  
...  
for each  $M(v_0, \dots, v_k)$  in tests  
  run  $x = M(v_0, \dots, v_k)$  and collect value  $x$   
  add " $x = M(v_0, \dots, v_k); \text{assert } x ==$ " +  $x$   
  to assert_tests  
return assert_tests
```

name  
or  
before

execute  
test  
&  
store test  
result

⊕ apply regression testing

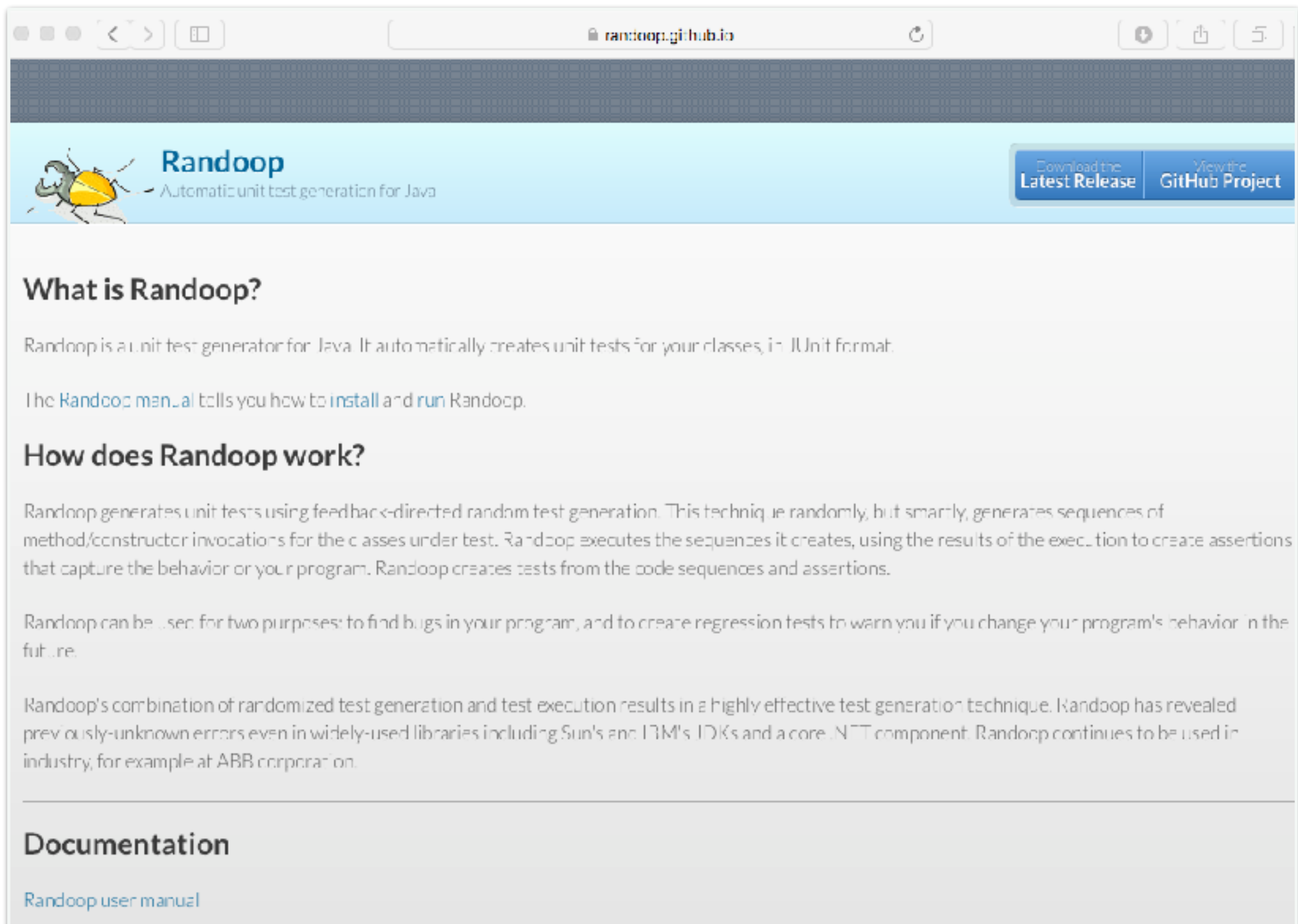
# Presupuesto de testing

↳ *oráculo (criterio de parada)*

- ¿Cuándo producimos la suficiente cantidad de tests?
  - Máxima cantidad de tests (ej: 1000 tests)
  - Tiempo máximo de generación (ej: 60 segundos)

What about objects with variable size?  
(dynamic data structure)

# Randoop (tool)



The screenshot shows the Randoop website in a web browser. The browser's address bar displays 'randoop.github.io'. The website has a light blue header with the Randoop logo (a yellow and black beetle) on the left, the text 'Randoop Automatic unit test generation for Java' in the center, and two buttons on the right: 'Download the Latest Release' and 'View the GitHub Project'. Below the header, the main content area has a light gray background. It starts with the heading 'What is Randoop?' followed by a paragraph explaining that Randoop is a unit test generator for Java. Below this is a link to the 'Randoop manual'. The next section is 'How does Randoop work?' followed by a paragraph describing the feedback-directed random test generation technique. Below that is another paragraph about the tool's purposes. The final section is 'Documentation' with a link to the 'Randoop user manual'.

**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 JNTT component. Randoop continues to be used in industry, for example at ABB corporation.

**Documentation**

[Randoop user manual](#)

# Randooop



- Stándard de-facto de **Random Testing** para el unit-level del lenguaje Java
- Genera automáticamente un conjunto de *test classes* **JUnit**
- Puede generar muchos tests en muy poco tiempo y exceder el límite de compilación

# 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

# Random Testing para programas Orientados a Objetos

```
class StackAr {  
    StackAr() {...}  
    StackAr(int) {...}  
    int size() {...}  
    boolean isEmpty() {...}  
    boolean isFull() {...}  
    void push(Object) {...}  
    Object pop() {...}  
    Object top() {...}  
}
```

how do  
we generate  
tests?

# Random Testing para programas Orientados a Objetos

- **Hasta ahora:** los programas bajo tests sólo tienen entradas primitivas (integers)
- ¿Qué hacemos si el parámetro es de tipo  $T$  (no es un tipo primitivo)?
  - Trivialmente: podemos elegir aleatoriamente entre:
    - El valor **null**
    - Invocar al constructor del tipo sin parámetros (si existe)

# 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 para programas Orientados a Objetos

- En los lenguajes orientados a objetos, necesitamos crear y modificar instancias de objetos
- Ejemplo:

```
// setup
LinkedList list0 = new LinkedList()
list0.add(null);
list0.add(null);
// exercise
list0.remove(0);
// check
assertEquals(1, list0.size());
```

# Random Testing para programas Orientados a Objetos

- En los lenguajes orientados a objetos, una excepción no siempre es síntoma de falla
- Ejemplo:

```
LinkedList list0 = new LinkedList()  
list0.get(-1); //throws IndexOutOfBoundsException
```

**Que tira una excepción no significa un error, ya que se supone que get() de un índice negativo debe emitir un IndexOutOfBoundsException**

# Random Testing para programas Orientados a Objetos

- Elegir valores aleatorios de tipos primitivos (floats, strings, integers, booleans es fácil)
  - Existe una representación finita (bits)
- ¿Cómo elegimos aleatoriamente una instancia de una clase (ej: HashSet, LinkedList, File)?
- Necesitamos poder crear **secuencias de llamados a métodos** para construir instancias complejas (i.e. interesantes)



# Catálogo de Métodos

- Necesitamos definir un **catálogo (menú)** de los métodos que podemos usar para poder crear instancias de objetos:
  - Por ejemplo:
    - `java.util.*`, `java.lang.*`
    - Todas las clases del proyecto
- Este **catálogo** puede ser calculado automáticamente

# Generación de Secuencias de Invocaciones (Tests)

```
while budget is not empty
  choose  $M(p_1:T_1, \dots, p_k:T_k):T_r$  from catalog  $C$ 
  for each input parameter of type  $p_i:T_i$ 
    choose randomly  $S_i$  from tests s.t. returns type  $T_i$ 

  build new sequence  $S_{new}=S_1; \dots; S_k; T_r$   $v_{new}=M(v_1, \dots, v_k)$ 
  Add  $S_{new}$  to tests

assert_tests := add_assertions(tests)
Return assert_tests
```

Algoritmo de generación random  
de tests.

catálogo

```
class Foo
  static int max(int a, int b) {...}

class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)

class Integer
  Integer(int)
  int intValue()
```

↓ invocaciones  
→ a los métodos de las variables  
secuencias

```
int int0 = 0;
```

```
boolean boolean0 = false;
```

```
Object object0 = new Object();
```

```
Integer integer0 = null;
```

```
float float0 = 0.0f;
```

```
LinkedList linkedList0 = null;
```

↳ randomly choose method to test



## catálogo

```
class Foo
  static int max(int a, int b) {...}

class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)

class Integer
  Integer(int)
  int intValue()
```

## secuencias

```
int int0 = 0;
```

```
boolean boolean0 = false;
```

```
Object object0 = new Object();
```

```
Integer integer0 = null;
```

```
float float0 = 0.0f;
```

```
LinkedList linkedList0 = null;
```

*add to list  
of references*

```
LinkedList linkedList0 = new LinkedList();
```

*→ randomly chosen*



## catálogo

```
class Foo
  static int max(int a, int b) {...}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

## secuencias

int int0 = 0;

boolean boolean0 = false;

Object object0 = new Object();

Integer integer0 = null;

float float0 = 0.0f;

LinkedList linkedList0 = null;

LinkedList linkedList0 = new LinkedList();

2<sup>nd</sup> random choice

int int0 = 0;  
LinkedList linkedList0 = new LinkedList();  
boolean boolean0 = linkedList0.remove(int0);

add



# catálogo

```
class Foo
```

```
    static int max(int a, int b) {...}
```

```
class java.util.LinkedList
```

```
    LinkedList()
```

```
    boolean add(Object)
```

```
    boolean remove(int)
```

```
    Object get(int)
```

```
class Integer
```

```
    Integer(int)
```

```
    int intValue()
```

# secuencias

```
int int0 = 0;
```

```
boolean boolean0 = false;
```

```
Object object0 = new Object();
```

```
Integer integer0 = null;
```

```
float float0 = 0.0f;
```

```
LinkedList linkedList0 = null;
```

```
LinkedList linkedList0 = new LinkedList();
```

```
int int0 = 0;
```

```
LinkedList linkedList0 = new LinkedList();
```

```
boolean b = linkedList0.remove(int0);
```

```
Object object0 = new Object();  
LinkedList linkedList0 = new LinkedList();  
boolean boolean0 = linkedList0.add(object0)
```

*random choice  
among the  
sequences that  
generate hint 5.*



# catálogo

```
class Foo
```

```
    static int max(int a, int b) {...}
```

```
class java.util.LinkedList
```

```
    LinkedList()
```

```
    boolean add(Object)
```

```
    boolean remove(int)
```

```
    Object get(int)
```

```
class Integer
```

```
    Integer(int)
```

```
    int intValue()
```

# secuencias

```
int int0 = 0;
```

```
boolean boolean0 = false;
```

```
Object object0 = new Object();
```

```
Integer integer0 = null;
```

```
float float0 = 0.0f;
```

```
LinkedList linkedList0 = null;
```

```
LinkedList linkedList0 = new LinkedList();
```

```
int int0 = 0;
```

```
LinkedList linkedList0 = new LinkedList();  
boolean b = linkedList0.remove(int0);
```

```
Object object0 = new Object();
```

```
int int0 = 0;
```

```
LinkedList linkedList0 = new LinkedList();
```

```
linkedList0.remove(int0);
```

```
boolean boolean0 = linkedList0.add(object0)
```

...



# Limitaciones (1)

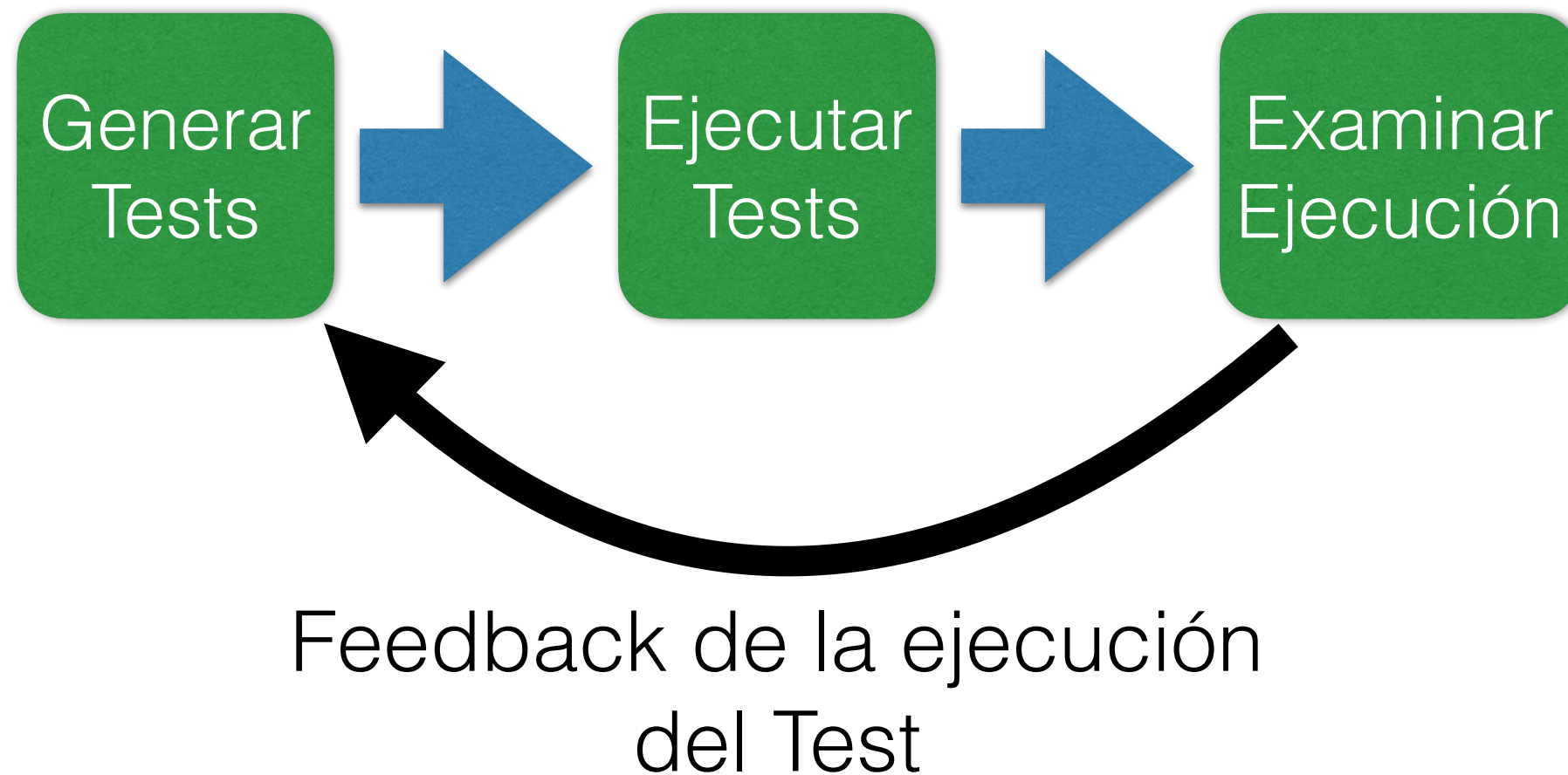
- Algunas secuencias no consiguen construir la instancia esperada.
- Ejemplo:

```
int int0 = -10;  
ArrayList list0 = new ArrayList(int0)
```

- Esta provoca una excepción
- Es usada para generar otras secuencias!

NO!  
→ si se genera una  
excepción ⇒ no es útil p/  
generar tests.

# Random Testing guiado por Feedback



modified pseudocode  
to filter sequences without  
exceptions.

**while** budget is not empty

**choose**  $M(p_1:T_1, \dots, p_k:T_k):T_r$  from catalog **C**

**for each** input parameter of type  $p_i:T_i$

**choose randomly**  $S_i$  from normal\_tests s.t. **returns**  $T_i$

**build** new sequence " $S_{\text{new}}=S_1; \dots; S_k; T_r$   $v_{\text{new}}=M(v_1, \dots, v_k)$ "

**Run**  $S_{\text{new}}$

**If** execution **signaled an exception**

**Add**  $S_{\text{new}}$  **to** exception\_tests

**Else**

**Add**  $S_{\text{new}}$  **to** normal\_tests

→ "flag" recursion  
de exceptions

**assert\_tests** := add\_assertions(normal\_tests+exception\_tests)

**Return** assert\_tests

# Limitaciones (2)

- Hay secuencias distintas que producen el mismo objeto

- Ejemplo:

*# sequences produce same object*

```
LinkedList list0 = new LinkedList()  
list0.isEmpty()
```

y

```
LinkedList list0 = new LinkedList()  
list0.add(null)  
list0.remove(0)  
list0.isEmpty()
```

↳ would be useful to define equivalence classes for sequences

**while** budget is not empty

**choose**  $M(p_1:T_1, \dots, p_k:T_k):T_r$  from catalog  $C$

**for each** input parameter of type  $p_i:T_i$

**choose randomly**  $S_i$  from normal\_tests s.t. **returns**  $T_i$

**build** new sequence " $S_{new}=S_1; \dots; S_k; T_r$   $v_{new}=M(v_1, \dots, v_k)$ "

**Run**  $S_{new}$

**If** execution **signaled an exception**

**Add**  $S_{new}$  **to** exception\_tests

**Else If** execution created a new instance

**Add**  $S_{new}$  **to** normal\_tests

**Else**

**Discard** sequence  $S_{new}$

**assert\_tests** := add\_assertions(normal\_tests+exception\_tests)

**Return** assert\_tests

→ didn't see  
the instance  
before.

# Oráculos

*randomness for oracles.*

- Si tenemos un oráculo automático, podemos clasificar los tests entre **passing** y **failing**
- Podemos usar los asserts dentro del programa (si es que siempre valen independientemente del valor)
- Podemos usar Oráculos Implícitos (e.g. propiedades generalmente válidas)

**while** budget is not empty

**choose**  $M(p_1:T_1, \dots, p_k:T_k):T_r$  from catalog **C**

**for each** input parameter of type  $p_i:T_i$

**choose randomly**  $S_i$  from normal\_tests s.t. **returns**  $T_i$

**build** new sequence " $S_{\text{new}}=S_1; \dots; S_k; T_r$   $v_{\text{new}}=M(v_1, \dots, v_k)$ "

**Run**  $S_{\text{new}}$

**If** execution **violated** a Oracle

**Add**  $S_{\text{new}}$  **to** failing\_tests

**Else If** execution **signaled** an exception

**Add**  $S_{\text{new}}$  **to** exception\_tests

**Else If** execution created a new instance

**Add**  $S_{\text{new}}$  **to** normal\_tests

**Else**

**Discard** sequence  $S_{\text{new}}$

passing\_tests := add\_assertions(normal\_tests+exception\_tests)

**Return** passing\_tests, failing\_tests



# 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...

# Randoop: Oráculos

oráculos implícitos  
en randoop.

- 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: <sup>other oracles</sup> Oráculos

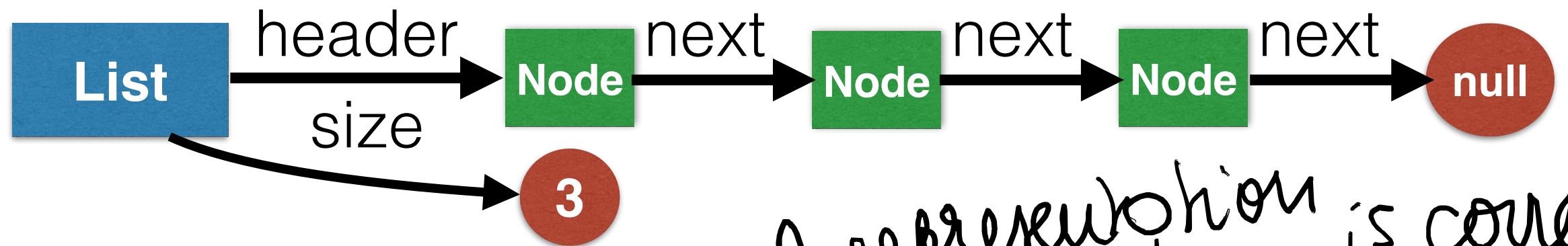
- Se produjo un *NullPointerException* cuando únicamente se utilizan argumentos que son distintos de null.
- Se produjo un *OutOfMemoryError*
- Se produjo un *AssertionFailure* (si *-ea* está activo)

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

# 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



→ true if representation is correct

@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;  
}
```

# Randooop



- Randooop 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)`



# Recap

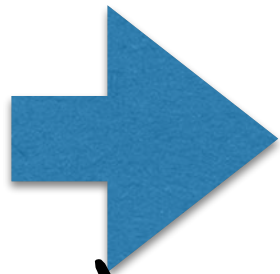
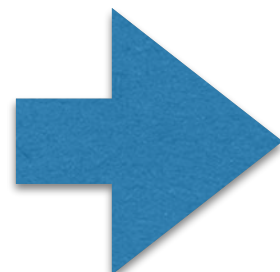
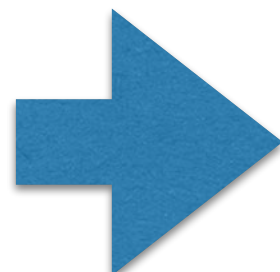
Programa+Catálogo

```
public static void Main()
{
    byte[] data = new byte[10];
    TcpClient server;
    try
    {
        server = new TcpClient();
    } catch (SocketException)
    {
        Console.WriteLine("Error");
    }
    return;
}
```

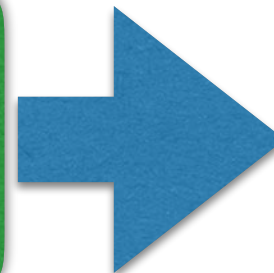
Budget  
(tiempo, #tests)

Oráculos

→ from checker  
or implicit.



Herramienta de  
Generación de  
Tests



# JUnit

Test Suite

PASS



FAIL



# Random Testing para lenguajes Orientados a Objetos

- Enfoque más popular: feedback-guided (randomloop)
- Evitar secuencias redundantes
- Evitar secuencias que no producen instancias
- Utilizar oráculos implícitos para clasificar los tests generados
- Necesitamos una condición de parada (tiempo, tests, etc.)

