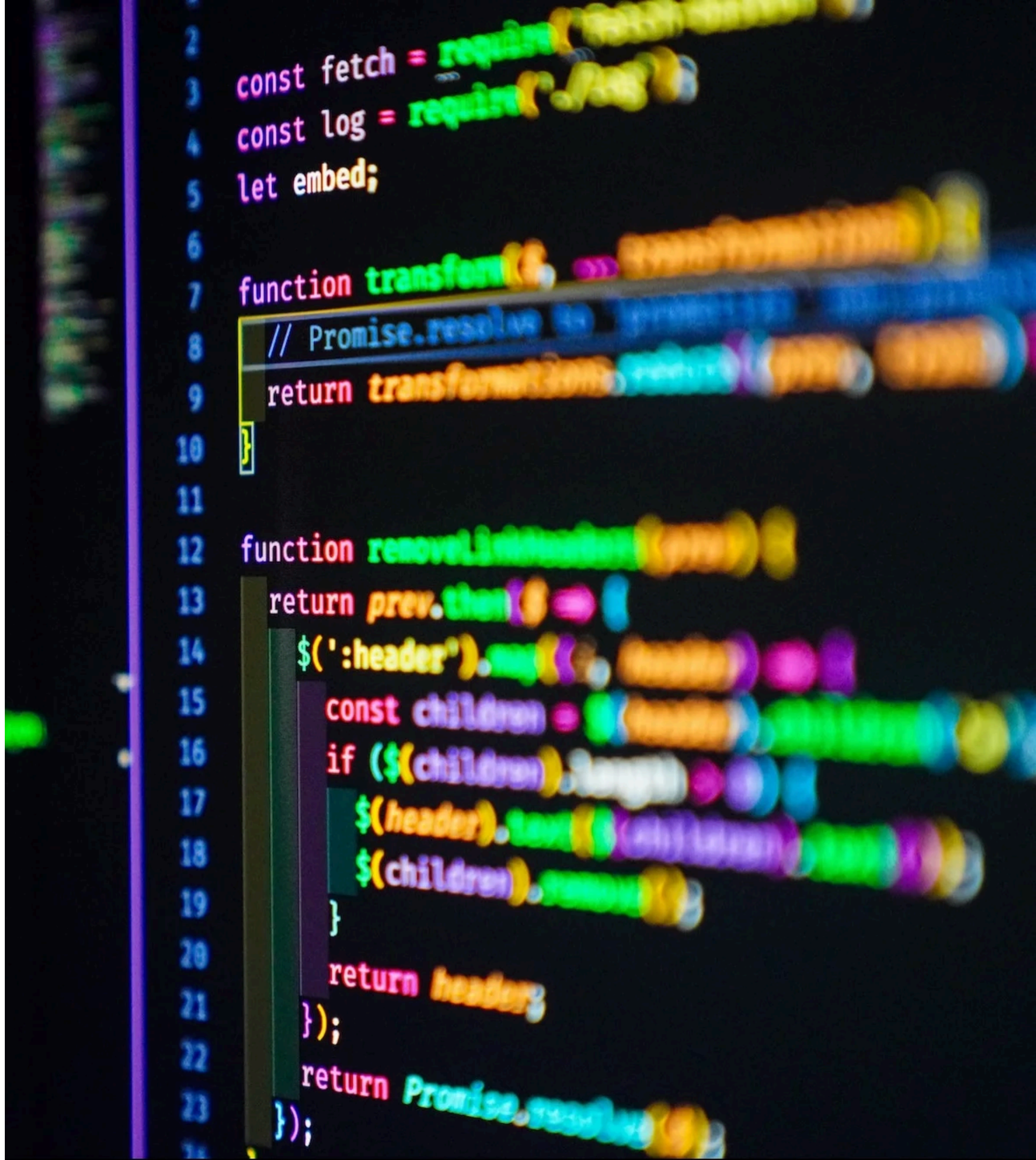
JCrasher

AN AUTOMATIC ROBUSTNESS TESTER FOR JAVA

Paper by: Christoph Csallner and Yannis Smaragdakis

Nuno Ribeiro 99293
Francisco Abreu 110946
Rúben Nobre 99321



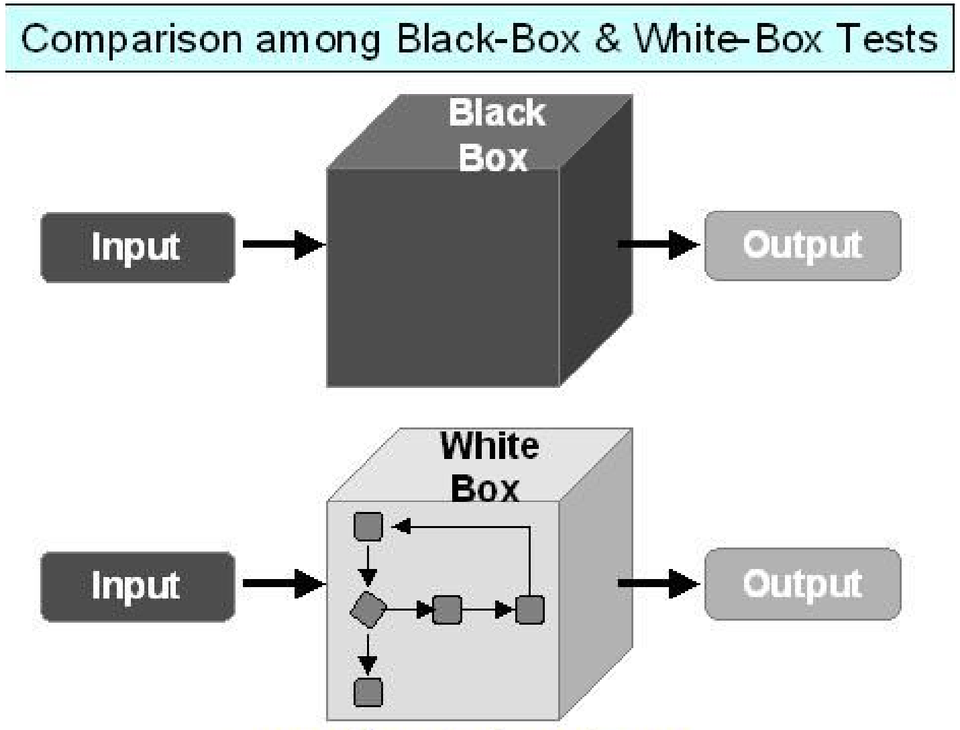
Introduction

What is JCrasher?

Robustness ——— Exceptions

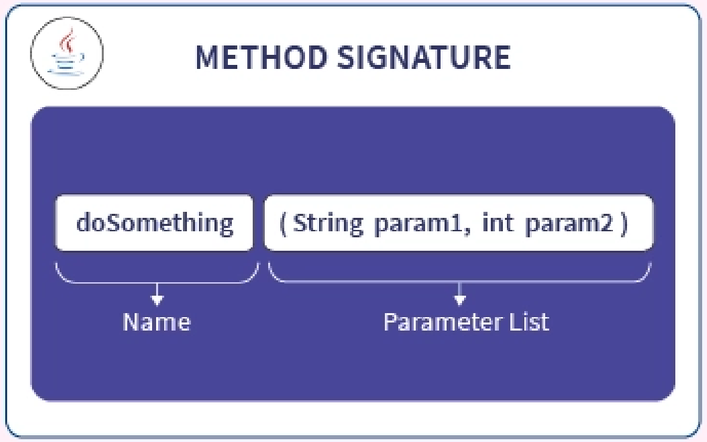
A class should never crash with an unexpected exception, no matter the input.

Previous Concepts



SRT

Randomized testing?



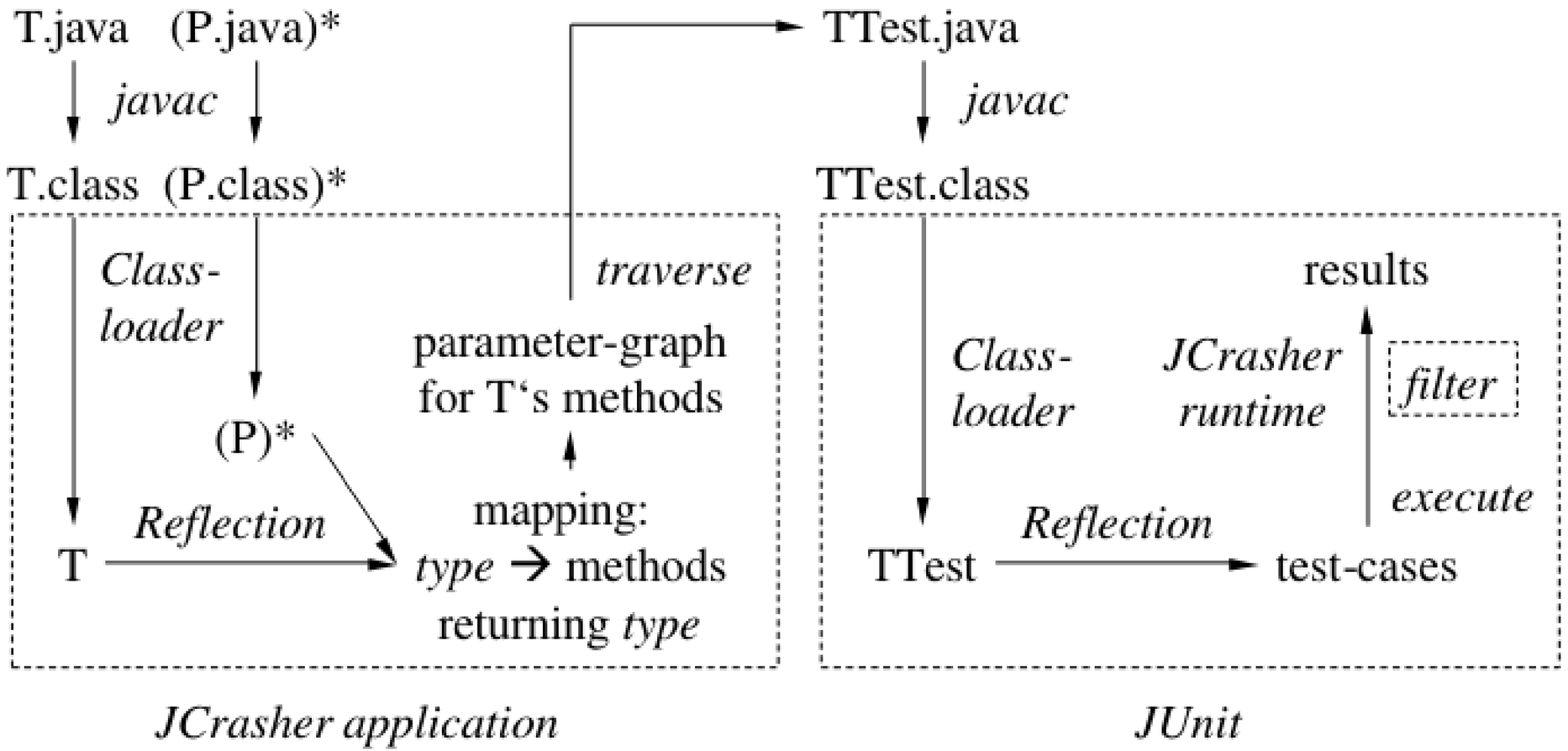
Pros vs Cons

Cheap

Good coverage

Reach unexpected scenarios

Illustrated



All features

Takes time into account

Clean slate

Uses heuristics

Easy to use

Test Case Generation

1. IDENTIFICATION OF TYPES & RESTRICTIONS

- Identify:
  - Types & restrictions

2. MAPPING OF METHODS AND VALUES

- Data structure in memory

3. DETERMINATION & WRITING OF TEST CASES

- Save as JUnit tests

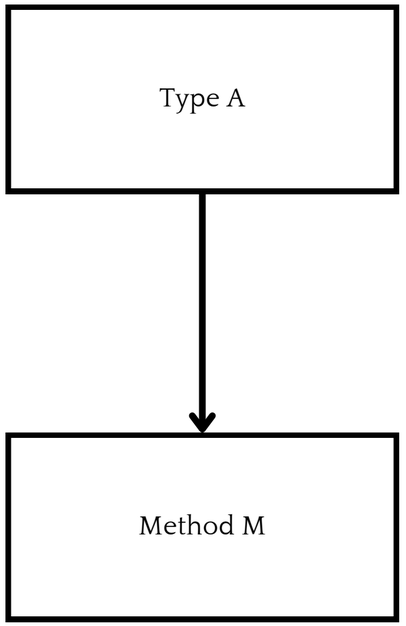
Parameter-graph

ANALYZES ...

- Method f of a class C w/ signature f(A1, A2, ..., AN) returns R

REGRA DE INFERÊNCIA

- R ← C, A1, A2, ..., AN



Parameter-graph

PARAMETERS

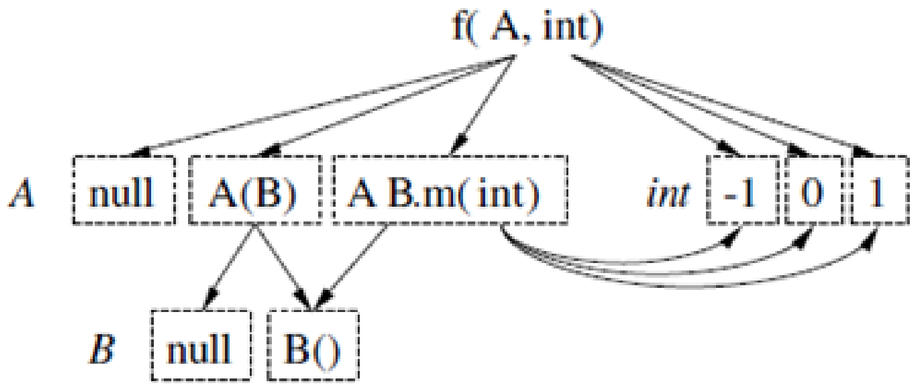
Type A & int

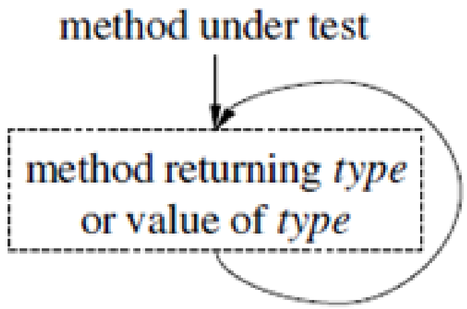
NODES

Values & types

EDGES

How to build a type using Inference Rules





Used to recursively generate parameters of the specified type

Example: A.m(int)

Parameter-graph Properties

DIVERSITY OF COMBINATIONS

- Different combinations w/ same values

FREQUENCY OF TESTS

- More parameter combinations, more frequent testing

FOCUS ON PARAMETERS

- Methods that do not return anything are excluded

Test Case Selection

Calculation of Sub-parameter Space

INITIAL VALUES

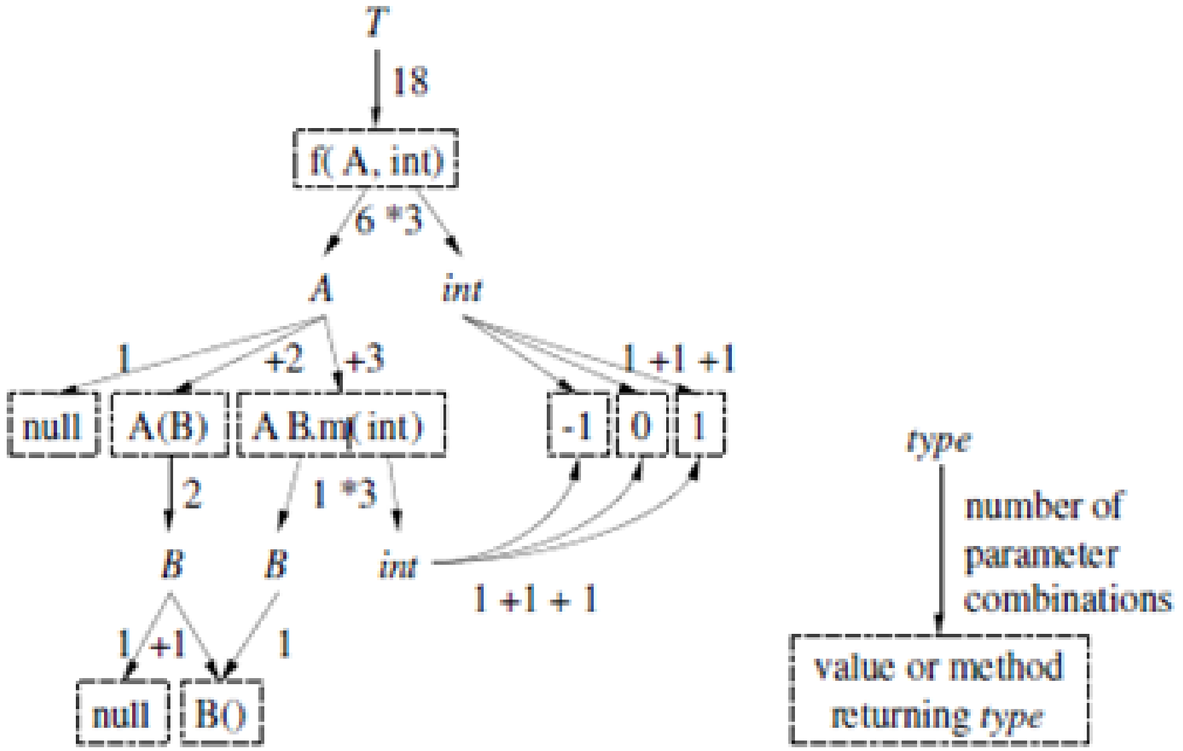
PRIMITIVE (1+1+1=3) & Type B (NULL+B()=2)

TYPE A

NULL + 1x2 + 1x3 = 1 + 2 + 3 = 6

BRANCHES COMBINATION

TYPE A x int = 6x3 = 18

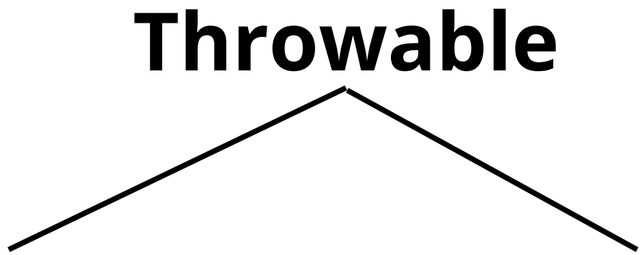


Test case execution

```
public void test1() throws Throwable{
    try{
        // test case
    }
    catch (Throwable e) {
        dispatchExcepiton(e);
    }
}
```

Each Exception thrown by a test case is caught and passed to the JCrasher runtime.

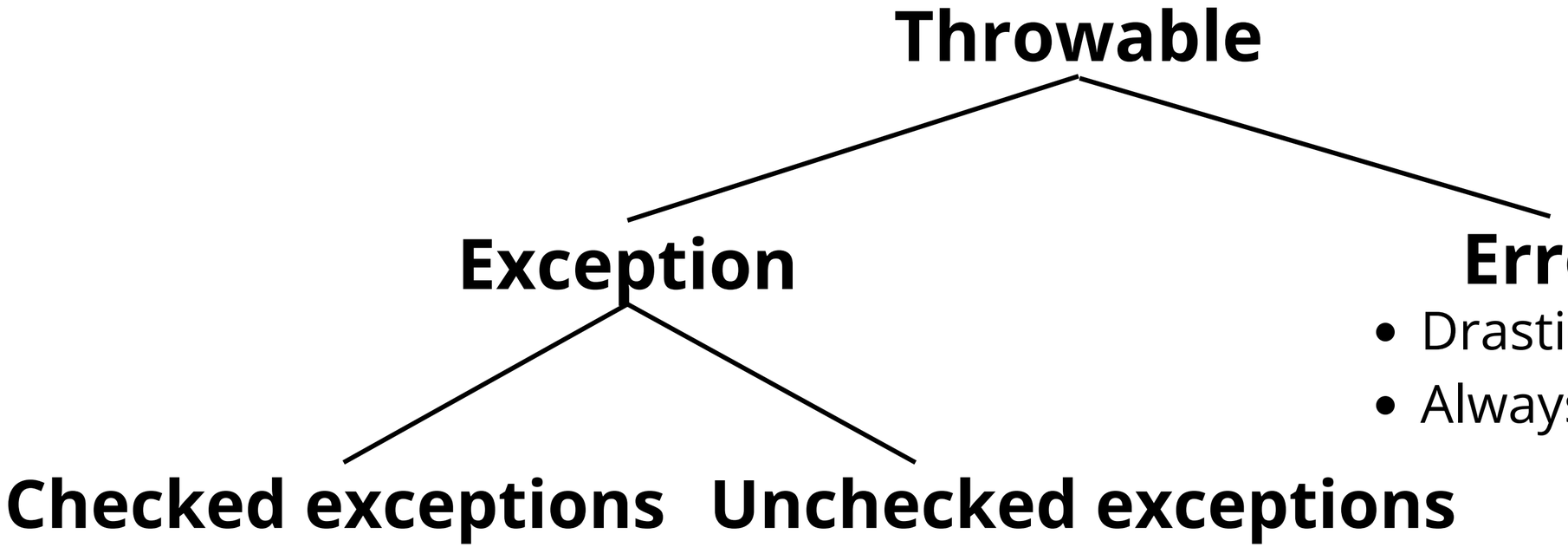
Exception Filtering



Error

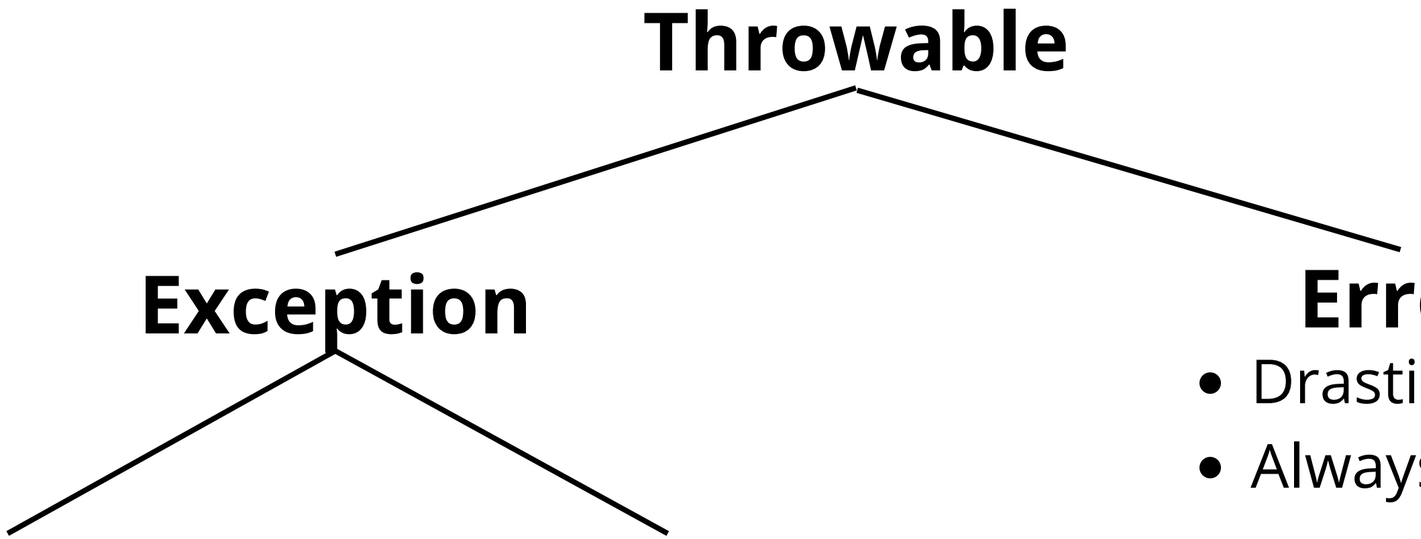
- Drastic problem
- Always passed to JUnit
- Ex: OutOfMemoryError

Exception Filtering



- Drastic problem
- Always passed to JUnit

Exception Filtering



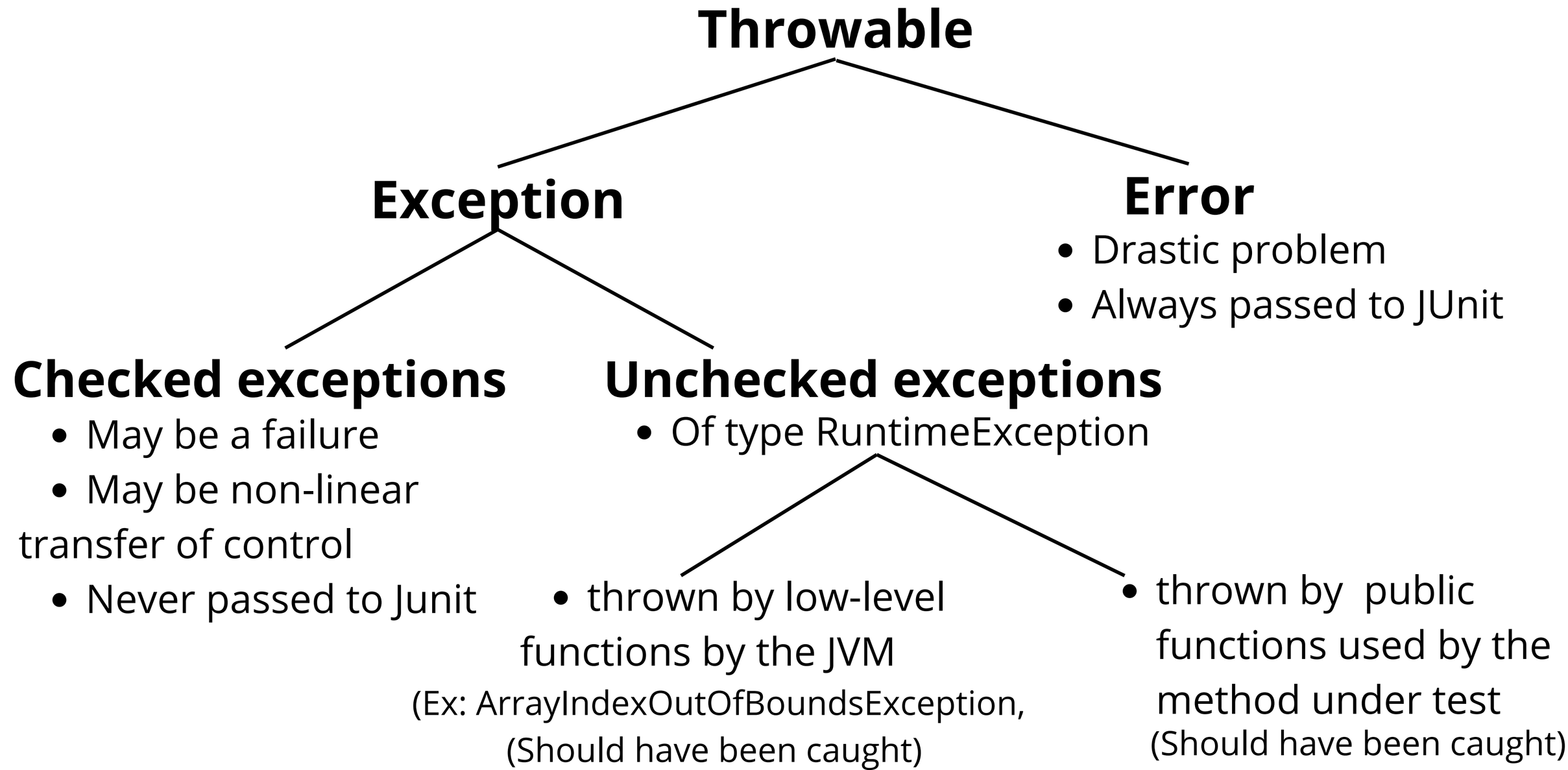
- Drastic problem
- Always passed to JUnit

Checked exceptions

- May be a failure
- May be non-linear transfer of control
- Never passed to JUnit

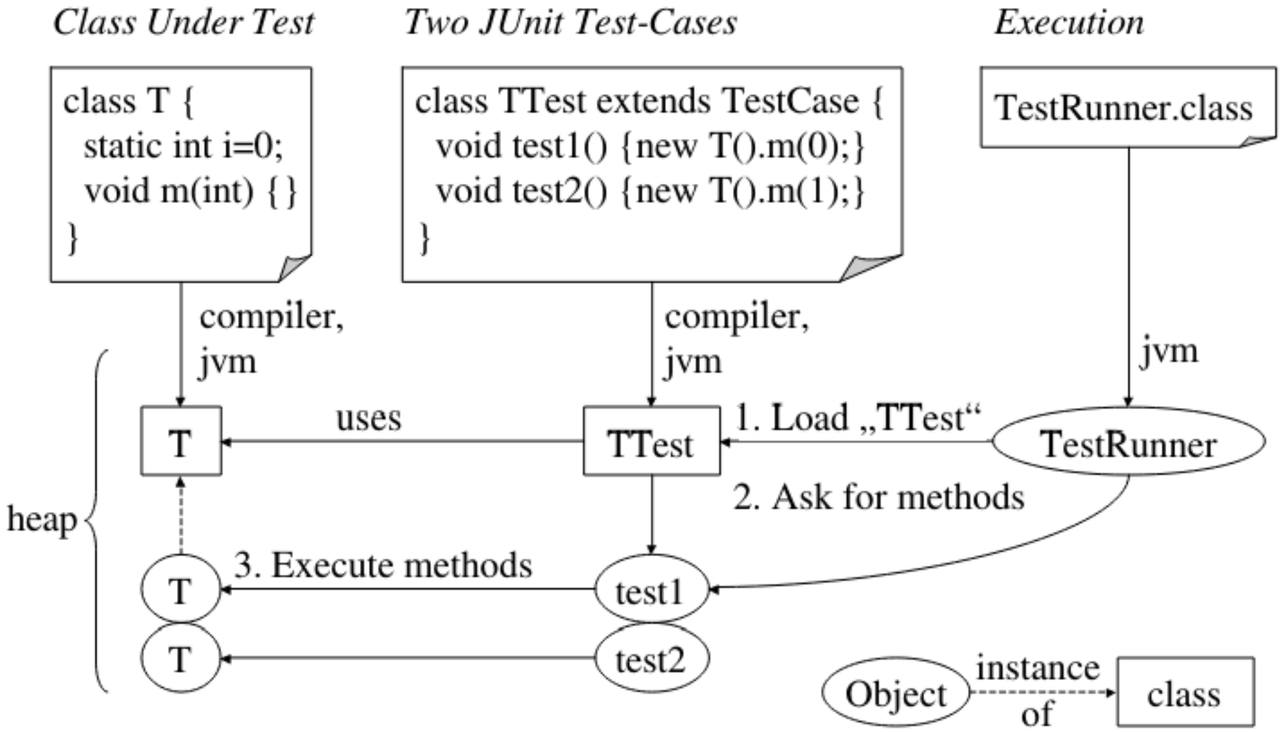
Unchecked exceptions

Exception Filtering



Resetting Static State

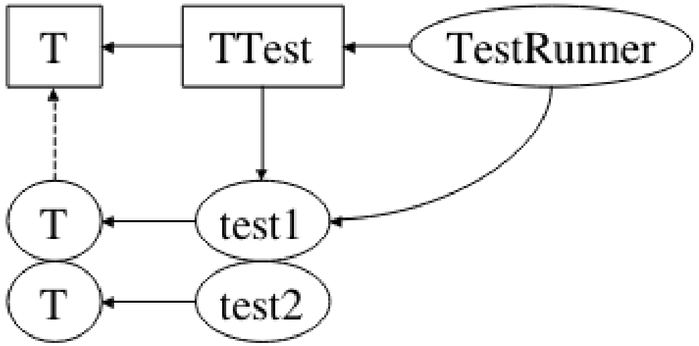
JUnit



Resetting Static State

JUnit

All test-cases test1(), test2() share the same environment, i.e. static state of T



Modified JUnit – junitMultiCL

Each test-case test1(), test2() has its own environment, i.e. class T

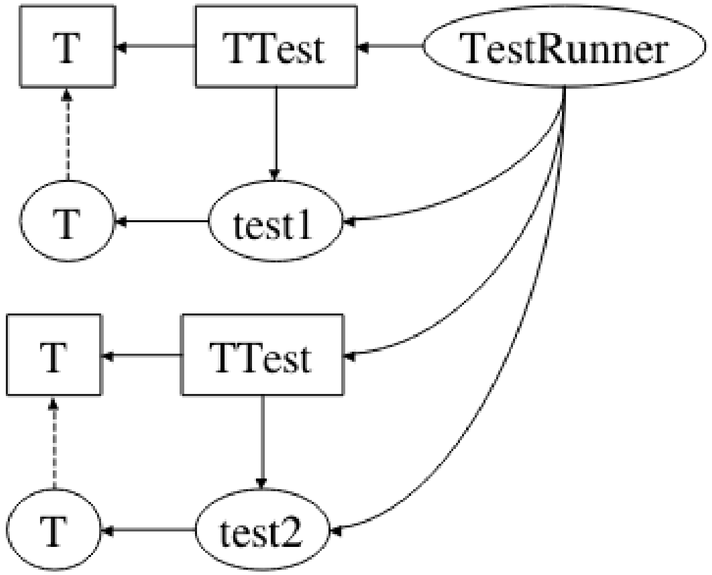


Figure 6. Concept of how to modify JUnit to undo changes of class state

RSS - Multiple Class Loaders

JUnit

1. One classloader loads all test-cases and all used classes
2. One suite-hierarchy is executed

Modified JUnit – junitMultiCL

1. Each test-case is loaded by a new classloader
2. The hierarchy is split up into a set of suites
3. The garbage collector can free the heap from executed suites

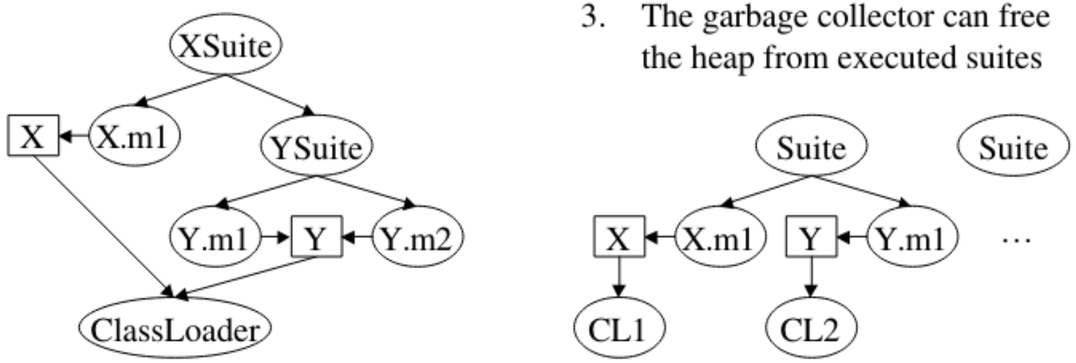


Figure 7. How JUnit's execution model is modified to undo changes to class state

Scalability:

Garbage collection

Memory consumption

100 << 100000

RSS - Load-time init

- Similar to JVM initialization
- Large improvement in scalability

<clinit> ➡ _clreinit()
_clinit()

avoid resetting final fields

RSS - Load-time init

- Reset happens at the end of each test
- Special treatment for *static* fields:

```
class A {static int a = B.b + 1;}
class B {static int b = A.a + 1;}
```

a = 2, b=1

a = 1, b= 2?

lazy vs eager loading

Performance

1. MULTIPLE CLASS LOADER APPROACH

- 2x faster
- Complete reset of state (safer)

2. RE-INITIALIZATION APPROACH

- 20x faster
- May not reset the static state

Future Directions

1. OBJECT-ORIENTED APPROACH

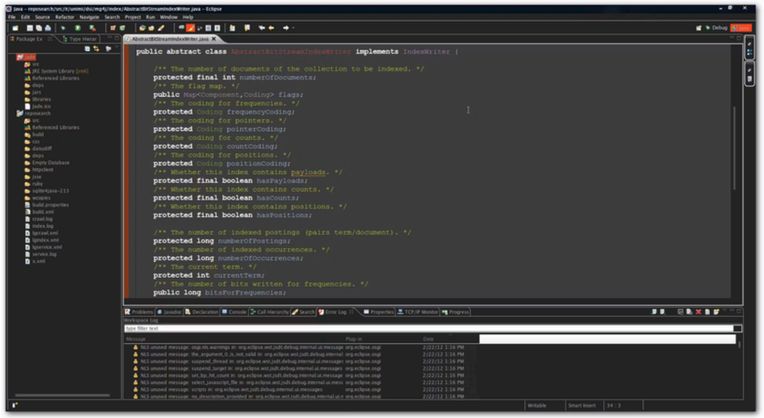
2. CORRECT BUT SLOWER APPROACH

3. OFFLINE CLASS REWRITING

JCrasher in Practice

Interactive Mode

- Good for newly written code
- Available through a plug-in in Eclipse IDE

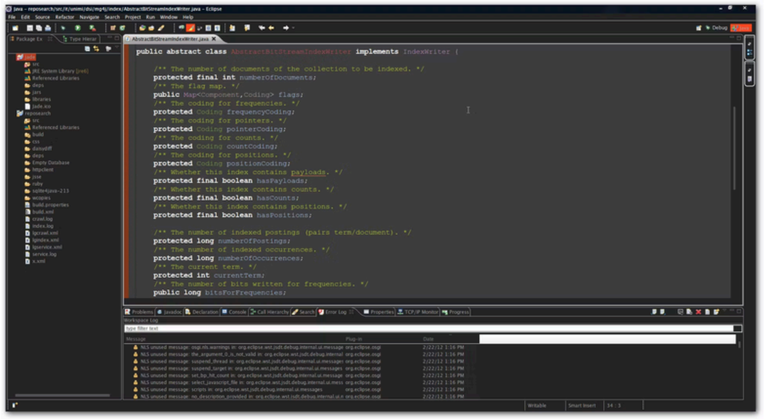


Batch Mode

JCrasher in Practice

Interactive Mode

- Good for newly written code
- Available through a plug-in in Eclipse IDE



Batch Mode

- Tests a big amount of classes and code
- Scalabilty Issues?

Scalability Problems

Five hour testing period → 1 000 000 test cases!

(200-300MB of code +
100-150MB of bytecode)

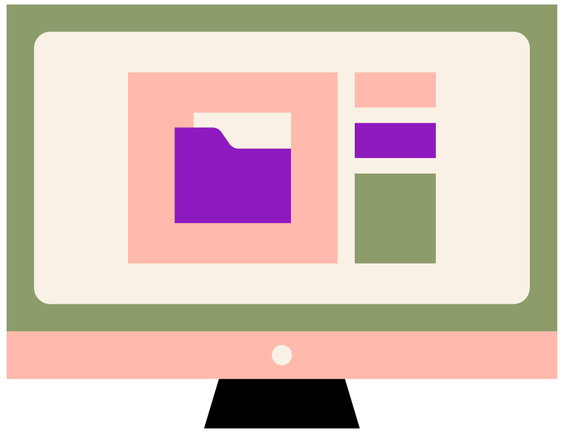
Compilation is expensive! → 7ms for each test case to compile

Scalability Problems

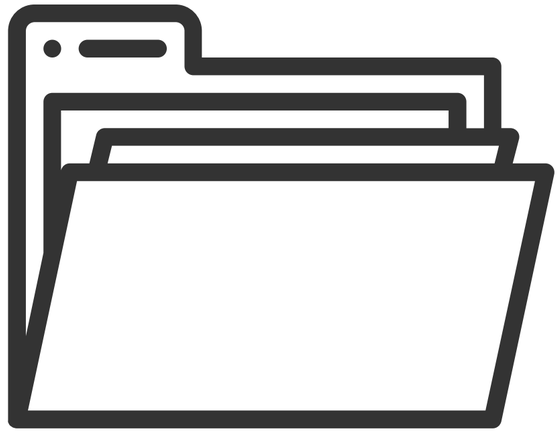
Is there a better way to do this?

JCrasher in Practice

Good for analyzing third party behaviour



Good for distuinguishing true errors from false positives



Results

Class name	Testee Author	Public methods	Test Cases	Crashes	Tests Problem reports	Redundant reports	Bugs
Canvas	SPEC	6	14382	12667	3	0	1?
P1	s1	16	104	8	3	0	1 (2?)
P1	s1139	15	95	27	4	0	0
P1	s2120	16	239	44	3	0	0
P1	s3426	18	116	26	4	0	1
P1	s8007	15	95	22	2	0	1
BSTree	s2251	24	2000	941	4	2	1
UB-Stack	Stotts	11	16	0	0	0	0

Results - Performance

Testee		Tests				
Class name	Author	Test Cases	Creation time [s]	Source size [kB]	Execution time [s]	Report size [kB]
Canvas	SPEC	14382	5.0	6000	14.9	30
P1	s1	104	0.3	20	1.0	1
P1	s1139	95	0.3	19	0.7	2
P1	s2120	239	0.3	55	1.3	2
P1	s3426	116	0.3	23	0.6	2
P1	s8007	95	0.3	19	0.5	1
BSTree	s2251	2000	0.9	564	3.4	6
UB-Stack	Stotts	16	0.3	4	0.5	0

Related Works

JUnit Test Class Generators

Enhanced Junit - also produces test classes, but not as many

Jtest - comercial tool, works automatically with white-box, but no clear exception division and less depth

Related Works

Providing Random Input

Ballista - focuses on assessing how effective exceptional input is handled by the software under test, crashing is always an error..

“Fuzz Testing of Application Reliability” project - has done black box testing using random input, crashing is always an error..

Claessen e Hughes - test pure functions in the Haskell programming language using random input.

Related Works

Formal Specifications

JML+JUnit work of Cheon and Leavens - the user has to suply the inputs after skeletal JUnit test case code is generated.

Korat - integrates with the JML+JUnit approach but automates the step of supplying test inputs, only non-isomorphic test cases are generated.

Conclusion

JCrasher is a a random testing tool for Java programs. Is able to discover arbitrary program bugs and it is ideal for robustness testing of public interfaces.