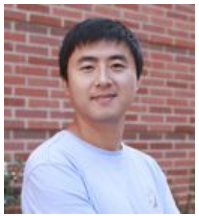
UCLA

JDEBLOAT: JAVA BYTECODE DE-BLOATING AND DE-LAYERING



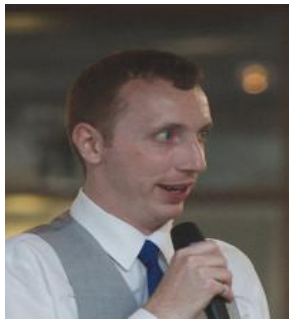


PI: MIRYUNG KIM, co-PIs: JENS PALSBERG, HARRY XU

J. EYOLFSON, U. HAMEED, C. KALHAUGE, H. MA, Y. QIAO, J. WANG, T. ZHANG

UCLA

TEAM INTRODUCTION



Instructor: Jon Eyolfson

Instructor: Christian Kalhauge

Instructor: Tianyi Zhang

TA: Yifan Qiao

TA: Jiyuan Wang

TA: Haoran Ma

TA: Usama Hameed

MOTIVATION FOR JAVA BYTECODE DEBLOATING

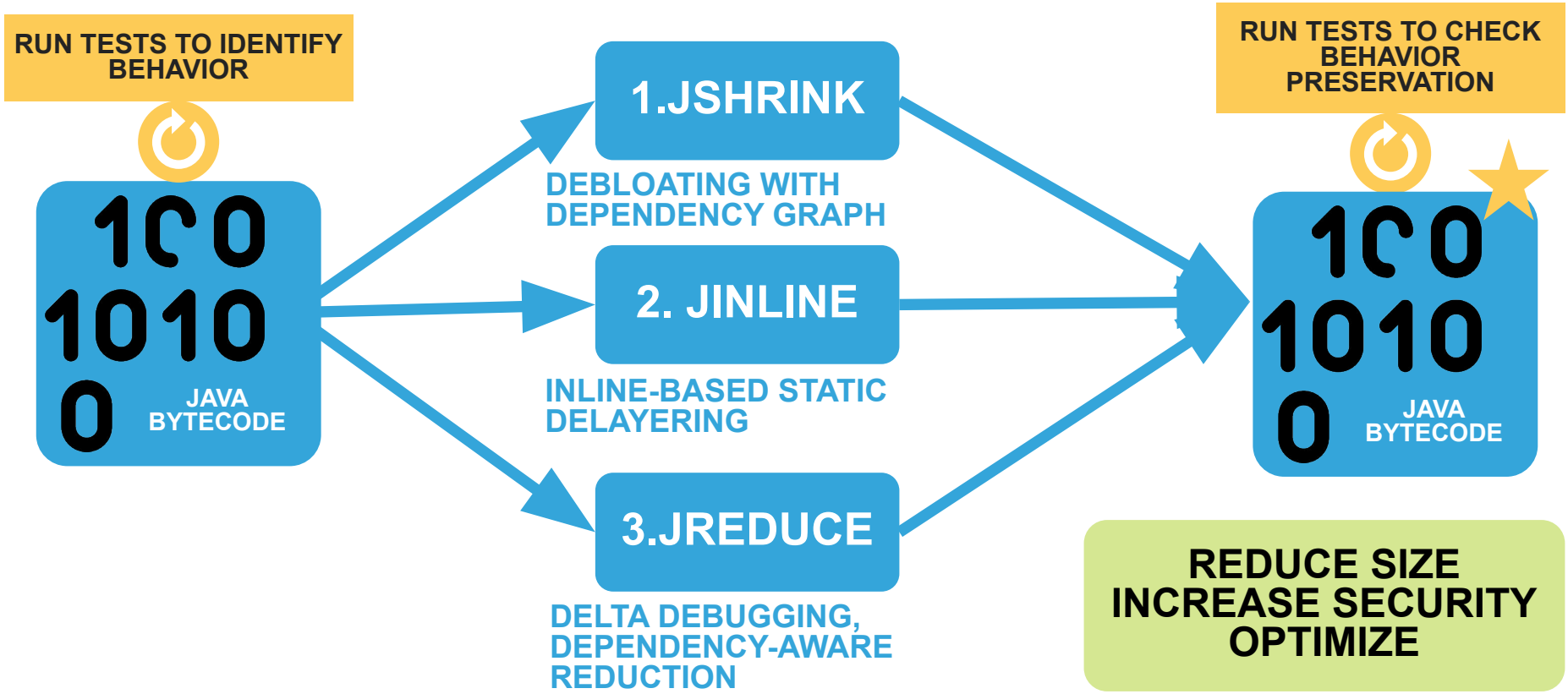
- Remove unused code and features
- Guarantee the same test execution behavior
- Reduce code size
- Remove security attack surfaces
- Optimize code

WHY IS JAVA DEBLOATING DIFFICULT?

- This is a long standing problem (e.g., JAX [Tip et al.], JRed [Jiang et al.])
- Commercial tools such as Proguard exist.
- Java is evolving and dynamic language features make debloating unsafe.
  - reflection, ambiguous refraction, dynamic class loading, dynamic proxy, invokedynamic (lambda expression), JNI, serialization, pluggable annotation

UCLA

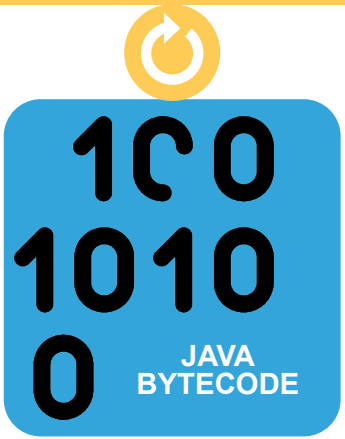
OUR THREE TOOLS



UCLA

OUR THREE TOOLS

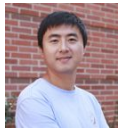
RUN TESTS TO IDENTIFY BEHAVIOR

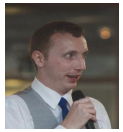


1. JSHRINK

DEBLOATING WITH DEPENDENCY GRAPH

2. JINLINE





INLINE-BASED STATIC DELAYERING

3. JREDUCE

DELTA DEBUGGING, DEPENDENCY-AWARE REDUCTION

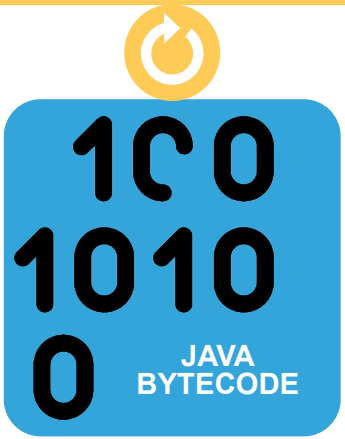
RUN TESTS TO CHECK BEHAVIOR PRESERVATION

REDUCE SIZE
INCREASE SECURITY
OPTIMIZE

UCLA

THREE TOOLS → A SINGLE INTEGRATED FRAMEWORK, JDEBLOAT

RUN TESTS TO IDENTIFY BEHAVIOR



JDEBLOAT

JREDUCE

DELTA DEBUGGING, DEPENDENCY-AWARE REDUCTION

JSHRINK

DEBLOATING WITH DEPENDENCY GRAPH

JINLINE

INLINE-BASED STATIC DELAYERING

RUN TESTS TO CHECK BEHAVIOR PRESERVATION

REDUCE SIZE
INCREASE SECURITY
OPTIMIZE

TUTORIAL WEBSITE

- The tutorial can be viewed from our tutorial website:

  http://debloating.cs.ucla.edu/debloat/SSSS20.html

- Section 1 refers to tools installation and setup
- Section 2 refers to hands-on maven-wrapper example.
- Section 3/4/5 refer to specific components in JDebloat: JInline, JShrink, and JReduce

For more details, refer to Section 1.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#11-Using-docker

Section 1.1: JDEBLOAT INSTALLATION

- Docker is a standalone container for all our software so you don't have to worry about dependencies and installing additional software
- (Optional) We provide a Docker image for our tools: http://debloating.cs.ucla.edu/dist/jdebloat_image.tgz
- (Optional) Load the image with: `docker load -i jdebloat_image.tgz`
- Run the image with: `docker run -it jdebloat`

For more details, refer to Section 1.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#11-Using-docker

Section 1.2: DIRECTORY LAYOUT

- All commands are from the root directory, look at the subdirectories with: `ls`
  - data - The benchmark folder. You can add new benchmarks to this folder.
  - output - All run commands output files here.
  - results - Summary results of the 25 default benchmarks
  - tools - Source code of all individual tools
- We automatically build Java projects using `mvn` and extract test cases

For more details, refer to Section 1.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#12-Directory-Layout

Section 1.3: PUSH BUTTON DEBLOATING

- Our main entry point is: `./jdebloat.py`
- Get a list of all our commands with: `./jdebloat.py -h`
- Setup your benchmarks and just use `./jdebloat.py`
  - --csv - Changes which csv file to use in the `data/` directory
  - --benchmark - Allows us to run an individual benchmark in the csv file

For more details, refer to Section 1.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#13-Help-and-Setup

Section 2.1: A PROGRAM TO BE DEBLOATED MUST HAVE TESTS

- All tools rely on tests to ensure they preserve behaviour
- We use the Surefire Report Plugin to get a list of test classes

- All test classes are in: `output/benchmark-id/initial/test.classes.txt`
- Exclude any test class by adding the class name to: `data/excluded-tests.txt`

Section 2.2: LET'S DEBLOAT "MAVEN-WRAPPER" USING JDEBLOAT

- Maven Wrapper is used for projects that need a specific version of Maven
- Create `data/tutorial.csv` with the following lines:
  - `id,url,rev`
  - `ssss,https://github.com/takari/maven-wrapper,2528f4144d0e50f1e10d7e84c1fddd1edf88ce58`
- Run it with: `./jdebloat.py run --csv=tutorial`

For more details, refer to Section 2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#2-Adding-a-Project

Section 2.2: DEBLOATING RESULTS: BEFORE & AFTER

- Each benchmark has a corresponding directory in output/benchmark-id, with the following subdirectories:
- benchmark — Source code for the benchmark.
- initial — Initial JAR files without modifications. initial+tool(s) — JAR files after running tool(s), in order.
- Each subdirectory with JAR files contains a stats.csv file with size statistic

1. JSHRINK

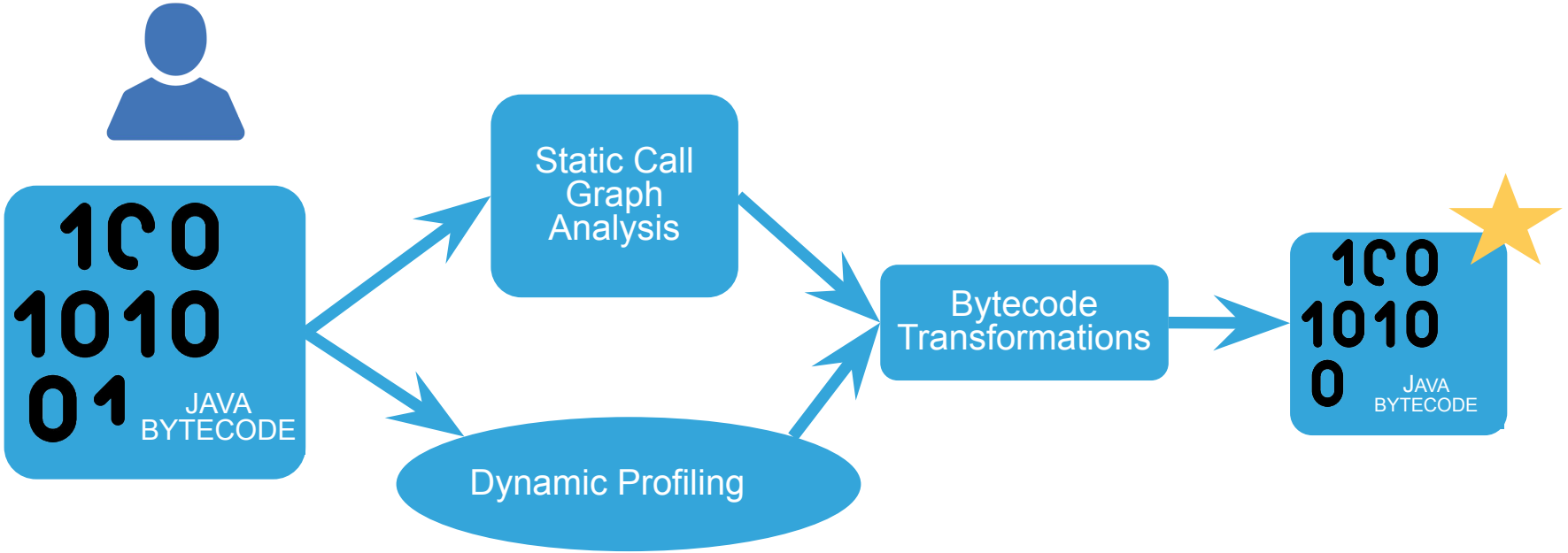
DEBLOATING WITH DEPENDENCY GRAPH AND DYNAMIC PROFILING

Instructor: Tianyi Zhang

Tianyi obtained his PhD in Computer Science from UCLA working with Prof. Miryung Kim. Now he is a Postdoctoral Fellow in Computer Science at Harvard University. He is working with Prof. Elena Glassman to design and build systems for interacting with population-level structures and patterns in large code and data corpora.

UCLA

1. JSHRINK: DE-BLOAT USING CALL GRAPH ANALYSIS & HANDLE DYNAMIC FEATURES



Section 3.1: DEBLOAT "MAVEN-WRAPPER" WITH JSHRINK

- Run the following command:
  - ./jdebloat.py run jshrink --csv=tutorial
- The JDebloat framework will only run JShrink without the other two tools.

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

Section 3.1: DEBLOAT "MAVEN-WRAPPER" WITH JSHRINK

- The debloating process will take about 2 min to finish. Once finished, it will print out the debloating result as follows:

```
================================================================================
                            Debloating Stats:
================================================================================
Using debloating tool(s): jshrink
Benchmark ssss:
  Size before debloating: 91499
  Size after  debloating: 69419
  Size reduction: 24.13%
  Total test cases before debloating: 19
  Total test cases after  debloating: 19, (19 successes, 0 failures)
[docker@e7dc727e9cc1 ~]$ >
```

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics

- The Key Design Principle:

  Never Remove Code that is Likely to be Used!

- A thorough, hybrid approach:
  - Static Call Graph Analysis: over-approximate used code
  - Dynamic Profiling: identify code invoked by dynamic features that may be overlooked by static analysis

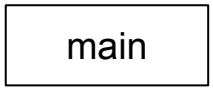
For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

UCLA

JShrink Basics: Static Call Graph Analysis

```
public class A {
  public A() {}
  public void foo() {
     bar();
  }
  public void bar() {
     print("abc");
  }
  public static void main(String[] args) {
     A a = new A();
     a.foo();
  }
}
```

Entry Point

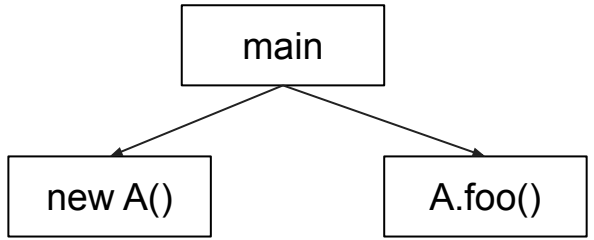


Static Call Graph Construction (1)

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics: Static Call Graph Analysis

```
public class A {
  public A() {}
  public void foo() {
     bar();
  }
  public void bar() {
     print("abc");
  }
  public static void main(String[] args) {
     A a = new A();
     a.foo();
  }
}
```

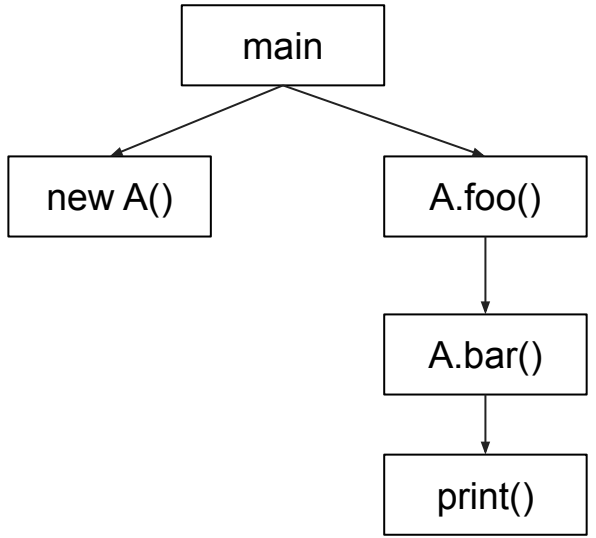


Static Call Graph Construction (2)

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics: Static Call Graph Analysis

```
public class A {
  public A() {}
  public void foo() {
     bar();
  }
  public void bar() {
     print("abc");
  }
  public static void main(String[] args) {
     A a = new A();
     a.foo();
  }
}
```

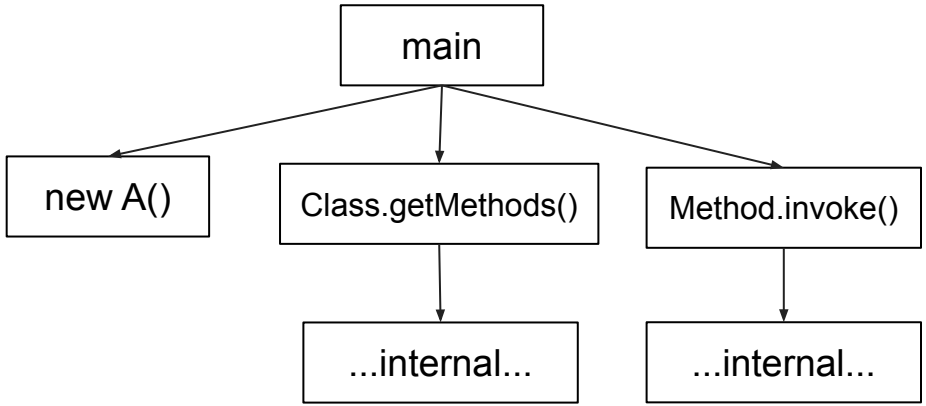


Static Call Graph Construction (3)

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics: Java Reflection

```
public class A {
  public A() {}
  public void foo() {
     bar();
  }
  public void bar() {
     print("abc");
  }
  public static void main(String[] args) {
     A a = new A();
     // a.foo()
     Method m = A.class.getMethods("foo");
     m.invoke(a);
  }
}
```

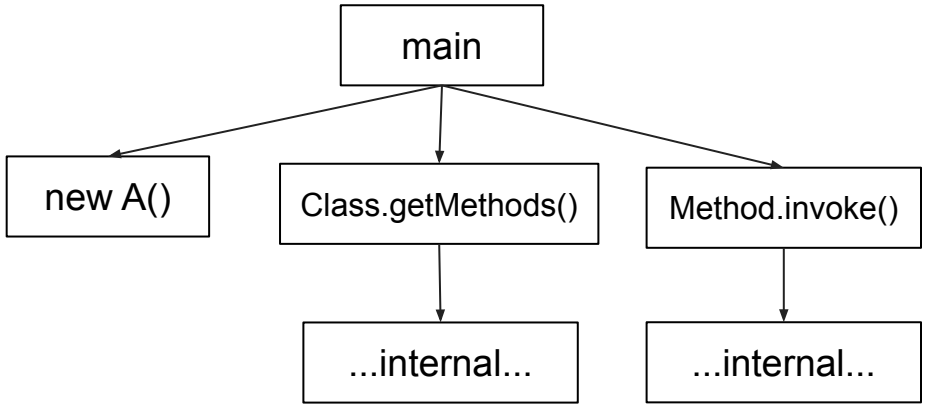


Static Call Graph

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics: Java Reflection

```
public class A {
  public A() {}
  public void foo() {
     bar();
  }
  public void bar() {
     print("abc");
  }
  public static void main(String[] args) {
     A a = new A();
     // a.foo()
     Method m = A.class.getMethods("foo");
     m.invoke(a);
  }
}
```



Static Call Graph

A.foo() and A.bar() are invoked at runtime but never appears in the static call graph!

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

JShrink Basics: Java Reflection

```
public class A {
  public A() {}
  public void foo() {
    bar();
  }
  public void bar() {
    print("abc");
  }
  public static void main(String[] args) {
    A a = new A();
    // a.foo()
    Method m = A.class.getMethods("foo");
    m.invoke(a);
  }
}
```

Call Graph Construction

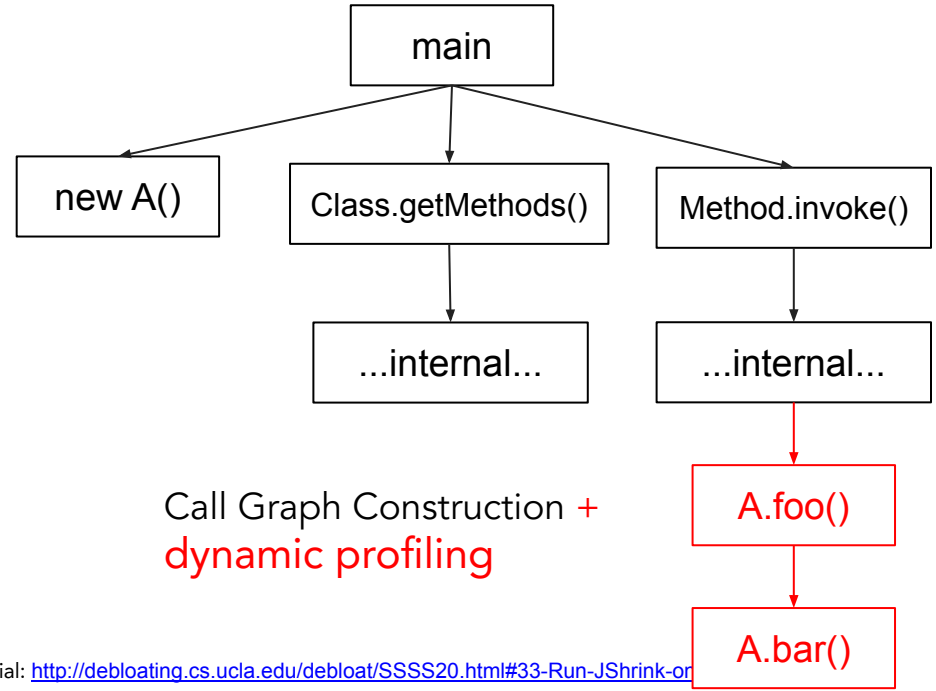
A.foo() and A.bar() are invoked at runtime but never appears in the static call graph!

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

UCLA

JShrink Basics: Java Reflection

- Handle Java Reflection with dynamic profiling
  - Use JVM TI APIs to instrument Java bytecode
  - Log execution traces



Call Graph Construction +
dynamic profiling

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-or

UCLA

JShrink Bytecode Debloating (A): Unused Method Removal

```
public class A {
  public void m1() {
     print("m1");
  }
  public void m2() {
     print("m2");
  }
  public static void main(String[] args) {
     A a = new A();
     a.m1();
  }
}
```


```
public class A {
  public void m1() {
     print("m1");
  }
  public void m2() {
     print("m2");
  }
  public static void main(String[] args) {
     A a = new A();
     a.m1();
  }
}
```

UCLA

JShrink Bytecode Debloating (B): Unused Field Removal

```
public class A {
  String a = "a";
  String b = "b";

  public void m1() {
     print(a);
  }
  public static void main(String[] args) {
     A a = new A();
     a.m1();
  }
}
```

UNUSED FIELD REMOVAL

```
public class A {
  String a = "a";
  String b = "b";

  public void m1() {
     print(a);
  }
  public static void main(String[] args) {
     A a = new A();
     a.m1();
  }
}
```

UCLA

JShrink Bytecode Debloating (C): Method Inlining

```
public class A {
  public void m1() {
    m2();
  }
  public void m2() {
    print("m2");
  }
  public static void main(String[] args) {
    A a = new A();
    a.m1();
  }
}
```

METHOD INLINING

```
public class A {
  public void m1() {
    m2();
    print("m2");
  }
  public void m2() {
    print("m2");
  }
  public static void main(String[] args) {
    A a = new A();
    a.m1();
  }
}
```

UCLA

JShrink Bytecode Debloating (D): Class Collapsing

```
public class A {
   m1();
}

public class B  extends A {
  m2();
}

public class C extends A {
   m3();
}
```

C is never used.

CLASS COLLAPSING

```
public class A {
   m1();
}

public class B  extends A {
   m1();
   m2();
}

public class C extends A {
   m3();
}
```

Other JShrink features to ensure debloating safety:

- Type dependency graph
- Checkpointing
- Handling new language features such as lambda expression
- Handling access control and class member visibility

JShrink: In-depth Investigation into Debloating Modern Java Applications.
Bruce, Zhang, Arora, Kim, Xu. FSE 2020. 12 pages.

Section 3.3: "HELLOWORLD" USING JSHRINK

- Look at a simple example present under examples/jshrink-test
- Run the following command to run JShrink on the example project:
  - ./jdebloat.py run jshrink --benchmark=jshrink-test

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

Section 3.2/3.3:
INSPECT DEBLOATED JAVA CLASSFILES

- JShrink and other tools in JDebloat debloat bytecode, not source code.
  - Source code of many software like Android Apps is often not available.
- (Option 1) Use javap to inspect Java bytecode.
  - javap -c A.class
- (Option 2) Use the disassemble.sh script to disassemble both input jar file and output jar file and vimdiff the outputs:
  - ./scripts/disassemble.sh jshrink jshrink-test

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

UCLA

Before Debloating

```
1  Compiled from "Main.java"
2  final class Main$1 implements java.util.Comparator<java.lang.Integer> {
3    Main$1();
4    public int compare(java.lang.Integer, java.lang.Integer);
5    public int compare(java.lang.Object, java.lang.Object);
6  }
7  Compiled from "Main.java"
8  public class Main {
9    public Main();
10   public static void main(java.lang.String[]);
11   static int access$000(java.lang.Integer, java.lang.Integer);
12 }
13 Compiled from "StandardStuff.java"
14 class StandardStuff$1 implements java.util.Comparator<java.lang.Integer> {
15   final StandardStuff this$0;
16   StandardStuff$1(StandardStuff);
17   public int compare(java.lang.Integer, java.lang.Integer);
18   public int compare(java.lang.Object, java.lang.Object);
19 }
20 Compiled from "StandardStuff.java"
21 class StandardStuff$NestedClass {
22   public void nestedClassMethod();
23   protected void nestedClassNeverTouched();
24   StandardStuff$NestedClass(StandardStuff$1);)
25 }
26 Compiled from "StandardStuff.java"
27 public class StandardStuff {
28   public StandardStuff();
29   protected void doNothing();
30   public java.lang.String getString();
31   public void publicAndTestedButUntouched();
32   public void publicAndTestedButUntouchedCallee();
33   public void publicNotTestedButUntouched();
34   public void publicNotTestedButUntouchedCallee();
35   protected void protectedAndUntouched();
36 }
37 Compiled from "StandardStuffSub.java"
38 public class StandardStuffSub extends StandardStuff {
39   public StandardStuffSub();
40   protected void protectedAndUntouched();
41   protected void subMethodUntouched();
42 }
```

After Debloating

```
1  Compiled from "Main.java"
2  final class Main$1 implements java.util.Comparator<java.lang.Integer> {
3    Main$1();

4    public int compare(java.lang.Object, java.lang.Object);
5  }
6  Compiled from "Main.java"
7  public class Main {
8    public Main();
9    public static void main(java.lang.String[]);
10   static int access$000(java.lang.Integer, java.lang.Integer);
11 }
12 Compiled from "StandardStuff.java"
13 class StandardStuff$1 implements java.util.Comparator<java.lang.Integer> {
14   final StandardStuff this$0;
15   StandardStuff$1(StandardStuff);

16   public int compare(java.lang.Object, java.lang.Object);
17 }
18 Compiled from "StandardStuff.java"
19 class StandardStuff$NestedClass {
20   public void nestedClassMethod();

21   StandardStuff$NestedClass(StandardStuff$1);)
22 }
23 Compiled from "StandardStuff.java"
24 public class StandardStuff {
25   public StandardStuff();
26   public void doNothing();
27   public java.lang.String getString();
28   public void publicAndTestedButUntouched();

29   public void publicNotTestedButUntouched();

30 }
31 Compiled from "StandardStuffSub.java"
32 public class StandardStuffSub extends StandardStuff {
33   public StandardStuffSub();

34 }
```

in our tutor

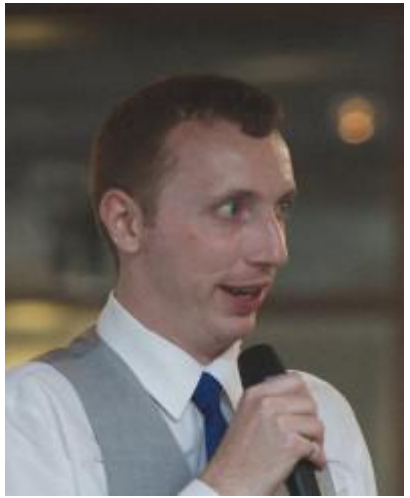
imple-Example

2. JINLINE

INLINE-BASED STATIC DELAYERING

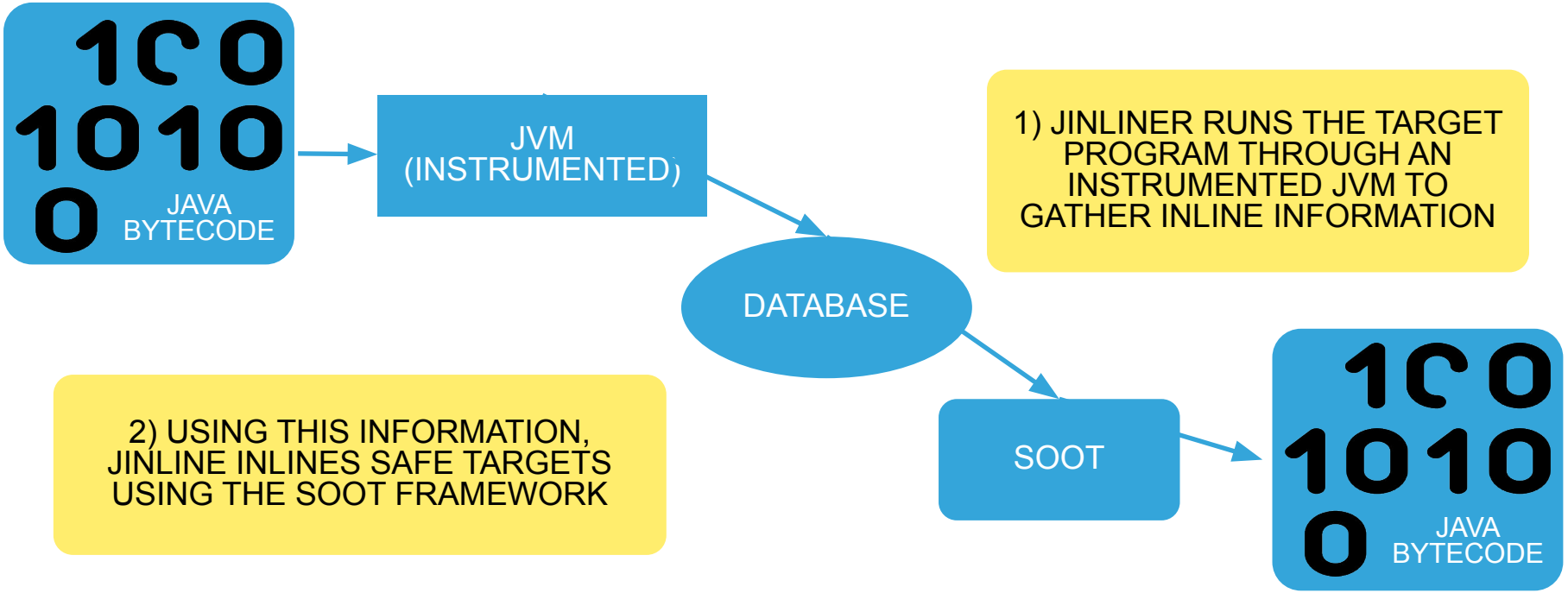
Instructor: Jon Eyolfson

Jon Eyolfson is a Postdoctoral scholar at the University of California, Los Angeles. His background is mix of static and dynamic analysis with empirical studies. His current interests are trying to leverage machine learning for software systems.



UCLA

2. JINLINE: INLINE-BASED STATIC DELAYERING



JInline Basics

- Dynamic Analysis Tool
- Uses User Written Test Cases to:
  - Exercise Callsites
  - Inlines “Important” Callsites
- Synergistically improve debloating with the other tools in the pipeline

UCLA

SECTION 4.1: "MAVENWRAPPER" USING JINLINE

- Run the following command:
  - ./jdebloat.py run jinline --csv=tutorial
- The JDebloat framework will only run JInline and print out the debloating result as follows:

```
========================================================================
                          Debloating Stats:
========================================================================
Using debloating tool(s): jinline
Benchmark ssss:
  Size before debloating: 91499
  Size after  debloating: 74941
  Size reduction: 18.10%
  Total test cases before debloating: 19
  Total test cases after  debloating: 19, (19 successes, 0 failures)
```

For more details, refer to Section 4.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#41-Run-JInline-on-Maven-wrapper

UCLA

JInline Basics: Inlining

```
    void printPersonName() {
        Student student = new Student();
        student.printName();
    }
```

```
class Student {
    void printName() {
        System.out.println("Hello student");
    }
}
```

UCLA

JInline Basics: Inlining

```
void printPersonName() {
    Student student = new Student();
    student.printName();
}
```

```
class Student {
    void printName() {
        System.out.println("Hello student");
    }
}
```

One possible Target Only

UCLA

JInline Basics: Inlining

```
void printPersonName() {
    Student student = new Student();
    System.out.println("Hello student");
}
```

```
class Student {
    void printName() {
        System.out.println("Hello student");
    }
}
```

One possible Target Only

UCLA

JInline Basics: Polymorphism

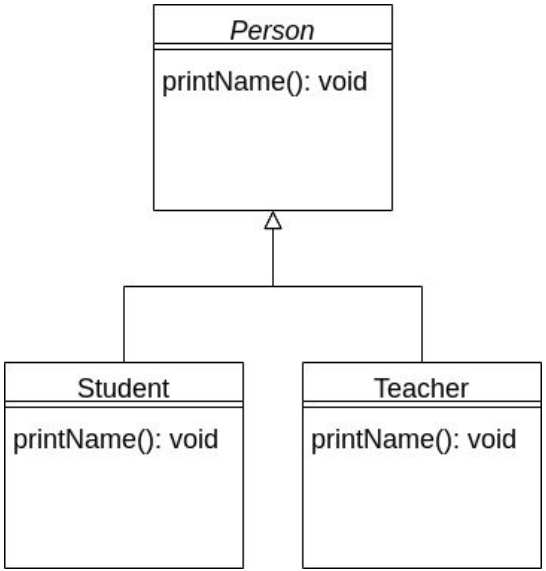
```
void printPersonName(bool cond) {
    Person person = null;
    if (cond)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

UCLA

JInline Basics: Polymorphism

```
void printPersonName(bool cond)
{
    Person person = null;
    if (cond)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

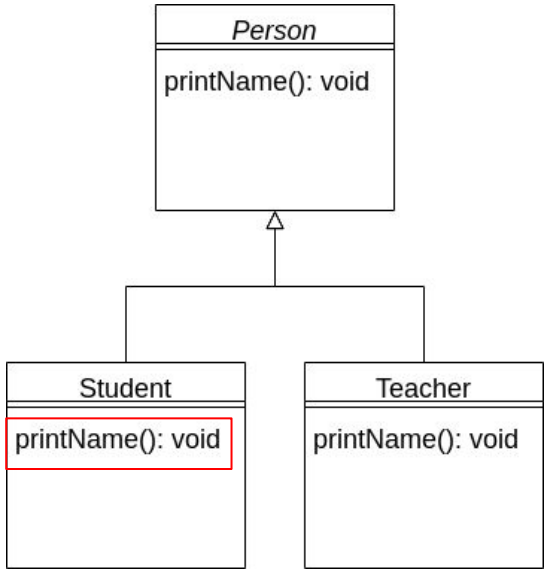
Class Hierarchy



JInline Basics: Polymorphism

```
void printPersonName(bool cond)
{
    Person person = null;
    if (true)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy

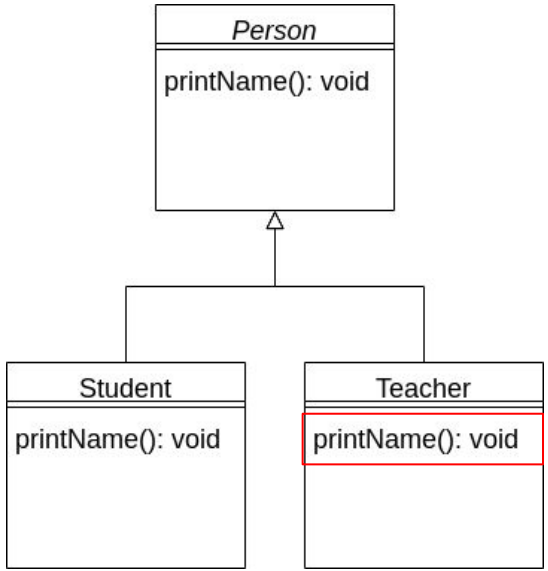


UCLA

JInline Basics: Polymorphism

```
void printPersonName(bool cond)
{
    Person person = null;
    if (false)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy

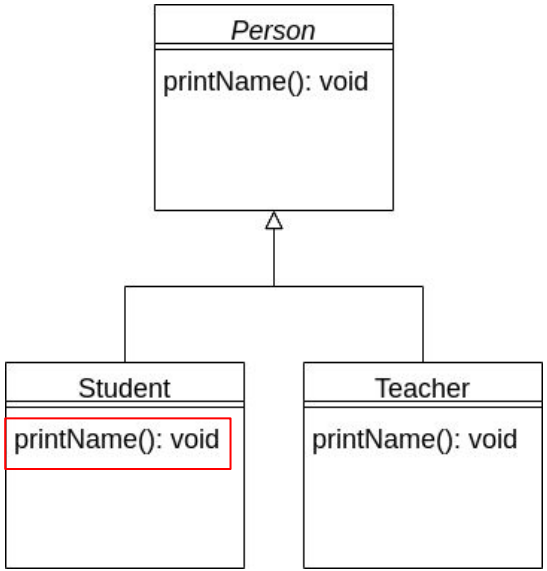


JInline Basics: Polymorphism

- Resolution Done at Runtime

```
void printPersonName(bool cond)
{
    Person person = null;
    if (true)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy

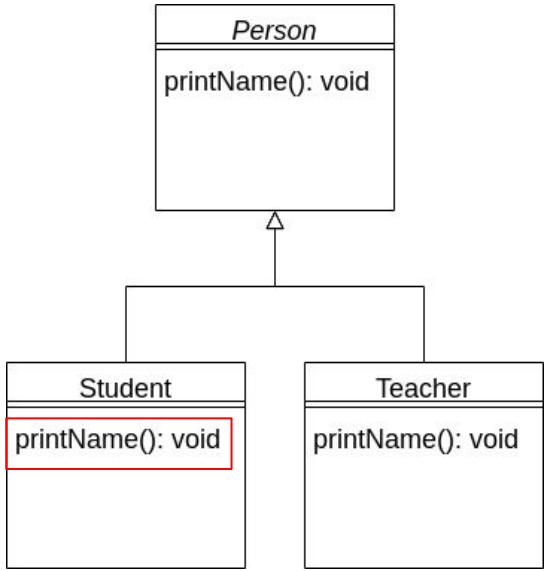


JInline Basics: Polymorphism

- Resolution Done at Runtime
- No way to statically debloat program

```
void printPersonName(bool cond)
{
    Person person = null;
    if (cond)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy



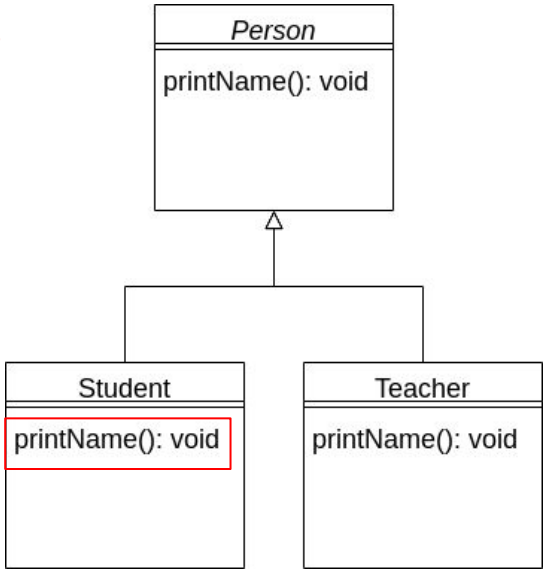
UCLA

JInline Basics: Polymorphism

- Resolution Done at Runtime
- No way to statically debloat program
- JInline exercises test cases to exploit inlining opportunities

```
void printPersonName(bool cond)
{
    Person person = null;
    if (cond)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy

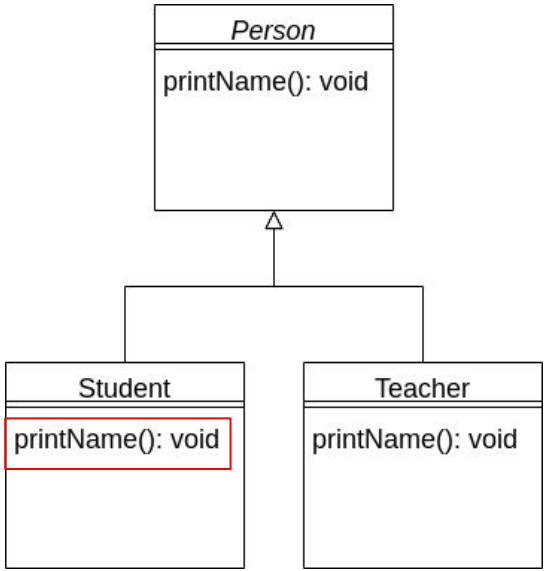


JInline Basics: Polymorphism

- Resolution Done at Runtime
- No way to statically debloat program
- JInline exercises test cases to exploit inlining opportunities

```
void printPersonName(bool cond)
{
    Person person = null;
    if (cond)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

Class Hierarchy



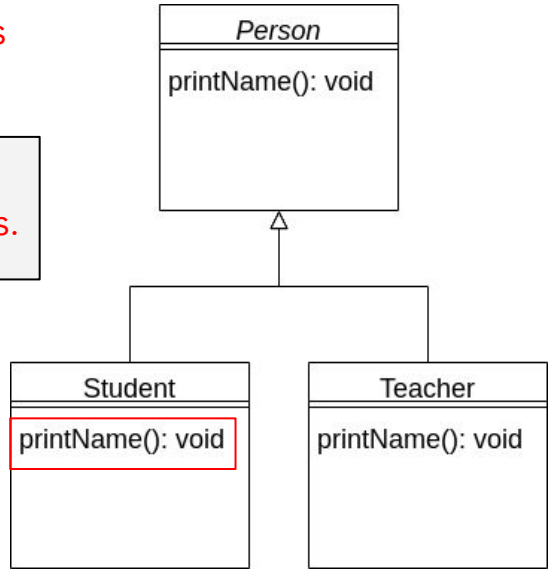
JInline Basics: Polymorphism

- Resolution Done at Runtime
- No way to statically debloat program
- JInline exercises test cases to exploit inlining opportunities

```
void printPersonName(bool cond)
{
    Person person = null;
    if (true)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

cond == true
In ALL test cases.

Class Hierarchy



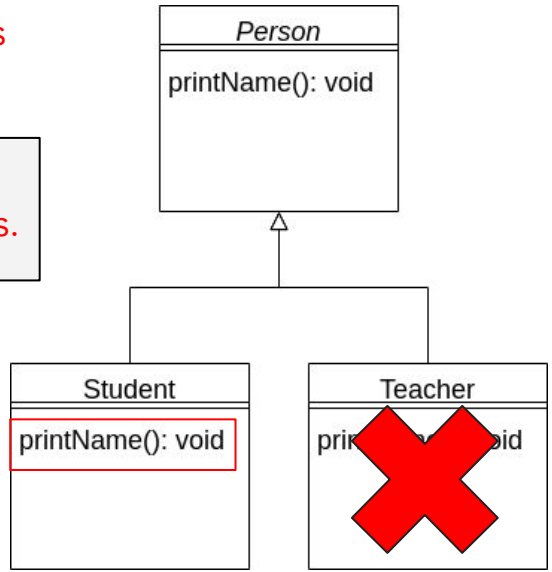
JInline Basics: Polymorphism

- Resolution Done at Runtime
- No way to statically debloat program
- JInline exercises test cases to exploit inlining opportunities

```
void printPersonName(bool cond)
{
    Person person = null;
    if (true)
        person = new Student();
    else
        person = new Teacher();

    person.printName();
}
```

cond == true
In ALL test cases.

Class Hierarchy



SECTION 4.2: "HELLOWORLD" USING JINLINE

- Simple example present under examples/jinline-test
- Two Source folders:
  - Application/src/main/
  - Library/src/main/java/edu/ucla/cs/onr/test/

For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

SECTION 4.2: "HELLOWORLD" USING JINLINE

Application/src/main/Application.java

```
public static int doubleOrSquare(int val,
boolean isDouble) {
    Doer doer = null;
    if (isDouble) {
        doer = new Doubler();
        ...
    } else {
        doer = new Squarer();
        ...
    }
    return doer.doIt(val);
}
```

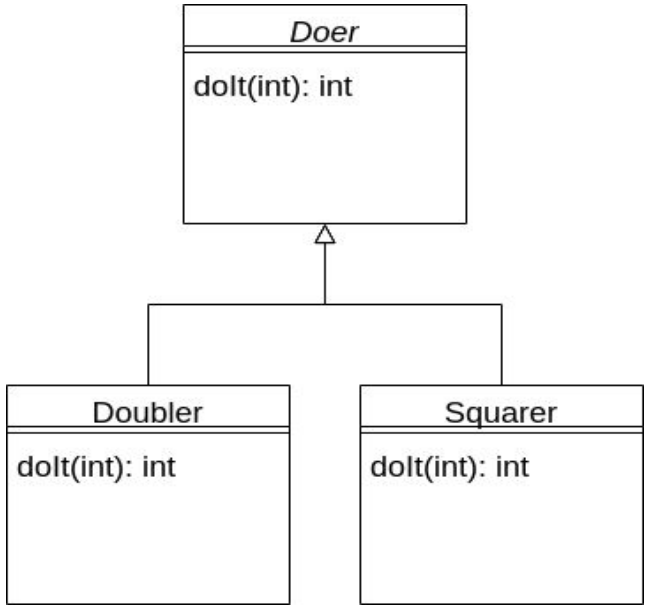
For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

UCLA

SECTION 4.2: "HELLOWORLD" USING JINLINE

Application/src/main/Application.java

```
public static int doubleOrSquare(int val,
boolean isDouble) {
    Doer doer = null;
    if (isDouble) {
        doer = new Doubler();
        ...
    } else {
        doer = new Squarer();
        ...
    }
    return doer.doIt(val);
}
```

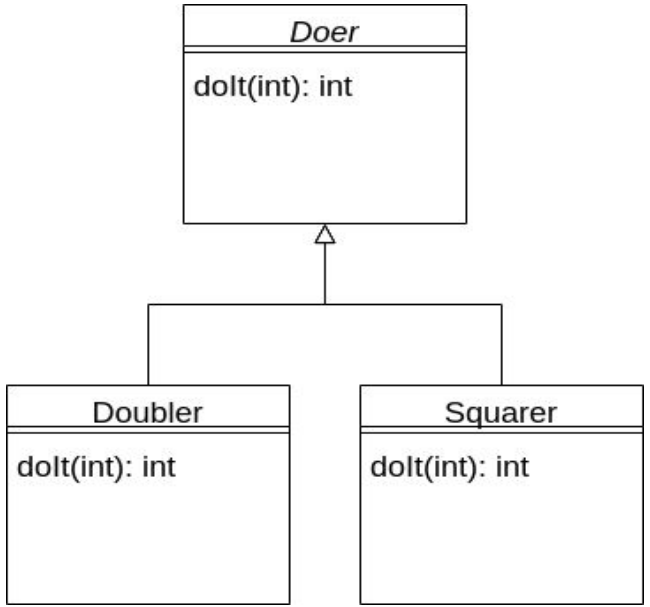


For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

SECTION 4.2: "HELLOWORLD" USING JINLINE

Application/src/main/Application.java

```
public static int doubleOrSquare(int val,
boolean isDouble) {
    Doer doer = null;
    if (isDouble) {
        doer = new Doubler();
        ...
    } else {
        doer = new Squarer();
        ...
    }
    return doer.doIt(val);
}
```



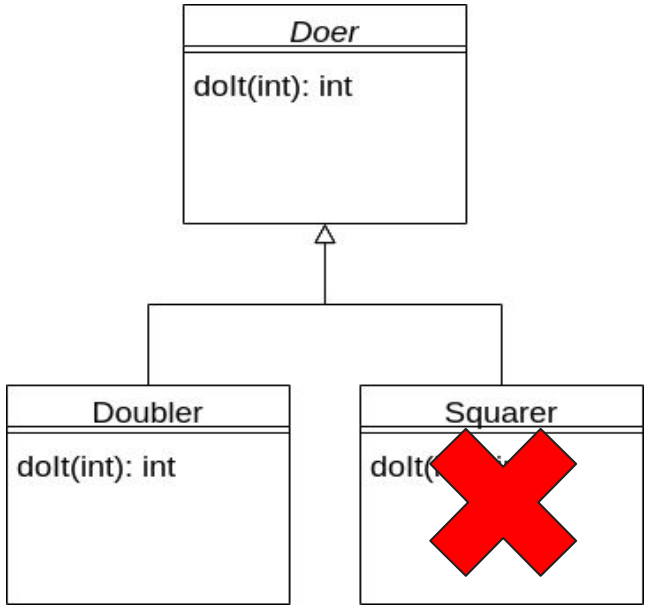
For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

SECTION 4.2: "HELLOWORLD" USING JINLINE

Application/src/main/Application.java

```
public static int doubleOrSquare(int val,
boolean isDouble) {
    Doer doer = null;
    if (true) {
        doer = new Doubler();
        ...
    } else {
        doer = new Squarer();
        ...
    }
    return doer.doIt(val);
}
```

isDouble == True
Exercised by
Testcase



For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

UCLA

SECTION 4.2: "HELLOWORLD" USING JINLINE

Before JInline

```
public static int doubleOrSquare(int,
boolean);
          ...
          ...
          64: iload_0
          65: invokevirtual
          68: istore_3
          69: iload_3
          70: ireturn
```

After JInline

```
public static int doubleOrSquare(int,
boolean);
          ...
          ...
          50: iconst_2
          51: imul
          52: istore_0
          53: goto 56
          56: iload_0
          57: ireturn
```

For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

UCLA

SECTION 4.2: "HELLOWORLD" USING JINLINE

Before JInline

```
public static int doubleOrSquare(int,
boolean);
          ...
          ...
          64: iload_0
          65: invokevirtual
          68: istore_3
          69: iload_3
          70: ireturn
```

After JInline

```
public static int doubleOrSquare(int,
boolean);
          ...
          ...
          50: iconst_2
          51: imul
          52: istore_0
          53: goto 56
          56: iload_0
          57: ireturn
```

For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

SECTION 4.2: "HELLOWORLD" USING JINLINE

- Run ./jdebloat.py run jinline --benchmark=jinline-test
- Output in output/jinline-test/initial+jinline
- To see the inlined callsite, follow the link at the bottom of this slide

For more details, refer to Section 4.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#42-Run-JInline-on-a-Simple-Example

3. JREDUCE

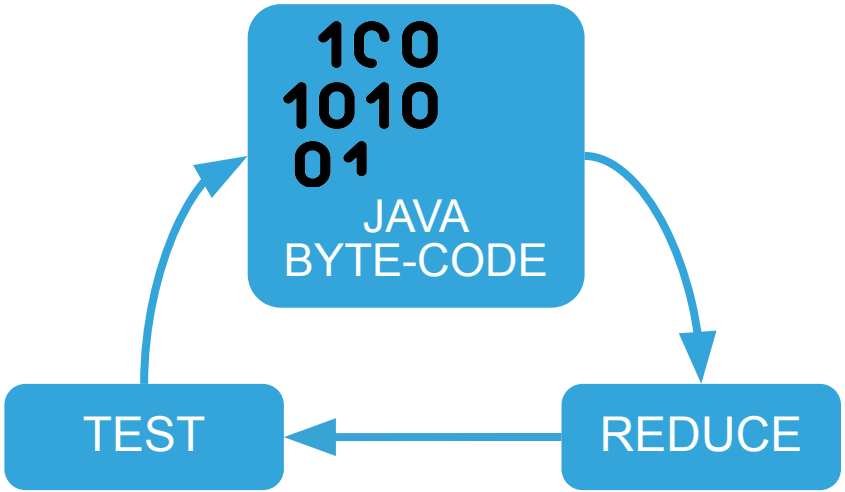
DELTA DEBUGGING, DEPENDENCY-AWARE REDUCTION

Instructor: Christian Kalhauge

Christian Kalhauge is a fifth-year PhD student at UCLA. His primary focus is on the interplay of static and dynamic analyses, bugs in concurrent programs, and automatic reporting bugs in tools that work on Java ByteCode.

3. JREDUCE: INPUT REDUCTION

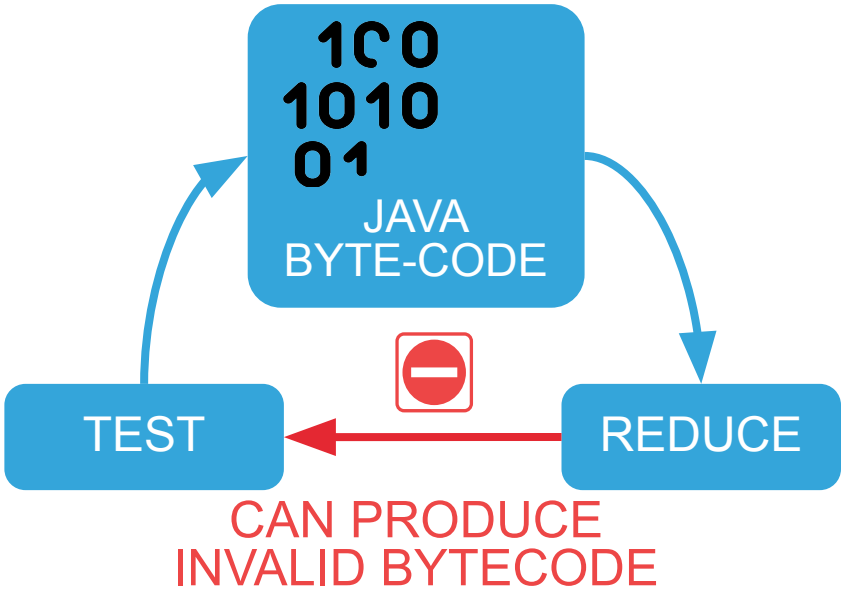
- Subtractive instead of Additive.
- Delta-Debugging as a Debloater: Consider the program as input to the test-cases.
- Remove classes while ensuring that the test-cases still succeed.



UCLA

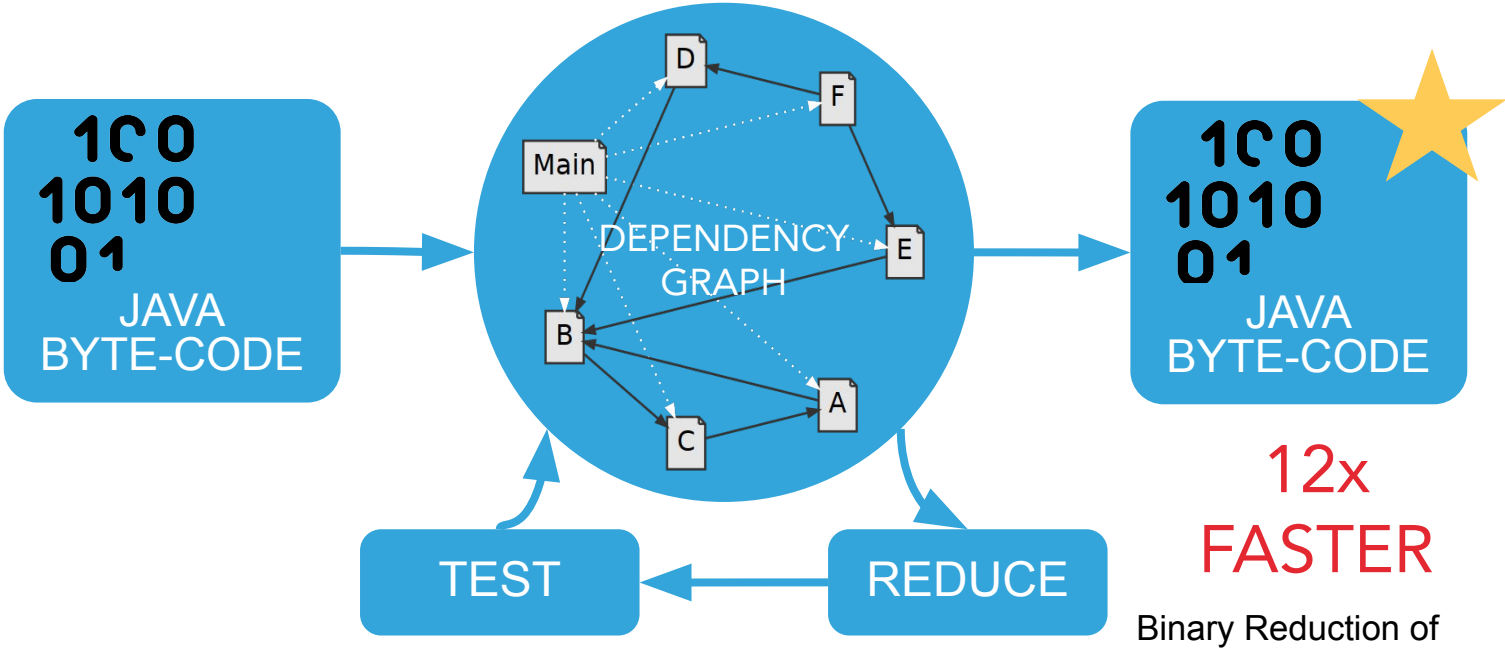
3. JREDUCE: INPUT REDUCTION

- Subtractive instead of Additive.
- Delta-Debugging as a Debloater: Consider the program as input to the test-cases.
- Remove classes while ensuring that the test-cases still succeed.
- Not all choices of classes are valid programs!



UCLA

3. JREDUCE: INPUT REDUCTION



12x FASTER

DOES NOT PRODUCE INVALID BYTECODE

Binary Reduction of Dependence Graphs, *Kalhauge and Palsberg* FSE 2019

SECTION 5.1: "MAVENWRAPPER" USING JREDUCE

- Run the following command:
  - ./jdebloat.py run jreduce --csv=tutorial
- The JDebloat framework will only run JReduce and print out the debloating result as follows:

```
=======================================================================
                        Debloating Stats:
=======================================================================
Using debloating tool(s): jreduce
Benchmark ssss:
  Size before debloating: 91499
  Size after  debloating: 30586
  Size reduction: 66.57%
  Total test cases before debloating: 19
  Total test cases after  debloating: 19, (19 successes, 0 failures)
```

For more details, refer to Section 5.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#51-Run-JReduce-on-Maven-wrapper

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- Simple example present under **examples/jreduce-test**
- The main code in **src/main/java/** contains seven classes:
  - A.java B.java C.java D.java E.java F.java Main.java

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- Simple example present under **examples/jreduce-test**
- The main code in **src/main/java/** contains seven classes:
  - A.java B.java C.java D.java E.java F.java Main.java
- Goal, to reduce the program while preserving the output:

```
$ java -cp target/classes Main A
Hello, A!
```

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- The problem: Main contain Reflection.

```
System.out.println(classloader.loadClass(args[0]).newInstance());
```

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- The problem: Main contain Reflection.

  System.out.println(classloader.loadClass(args[0]).newInstance());

- Running the program with another input print the toString of that Class:

```
$ java -cp target/classes Main D
Hello, D!
```

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- Input Reduction: Reduces an input program while maintaining the behaviour.

```
$ jreduce -o reduced target/classes/ -- java -cp {} Main A
...
```

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- Input Reduction: Reduces an input program while maintaining the behaviour.

```
$ jreduce -o reduced target/classes/ -- java -cp {} Main A
...
```

- It runs java -cp {} Main A 9 times to figure out the minimal program:

```
$ ls reduced
A.class  B.class  C.class  Main.class
```

```
$ java -cp reduced Main A
Hello, A!
```

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

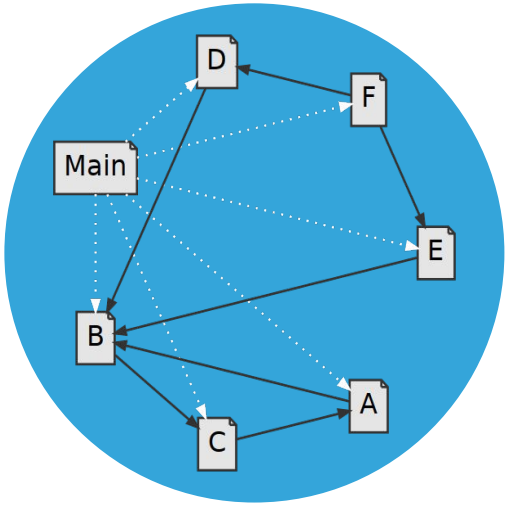
SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

- 9 iterations is fast! ddmin uses 22 to reduce to an invalid program!

For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

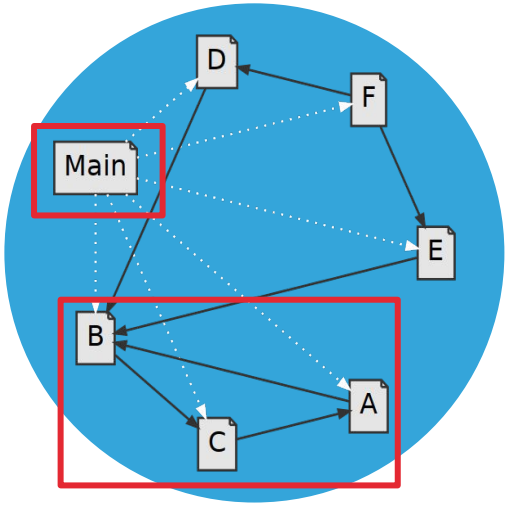
- 9 iterations is fast! ddmin uses 22 to reduce to an invalid program!
- We are faster because we model the validity of the program using a graph.



For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

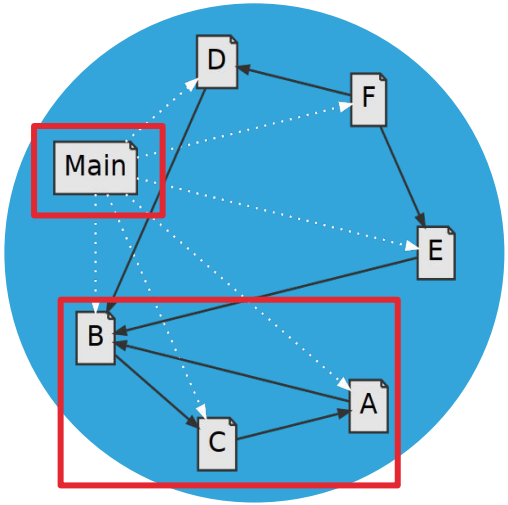
- 9 iterations is fast! ddmin uses 22 to reduce to an invalid program!
- We are faster because we model the validity of the program using a graph.
- Each closure is a valid program: our final program contains two.



For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

SECTION 5.2: "JREDUCE-TEST" USING JREDUCE

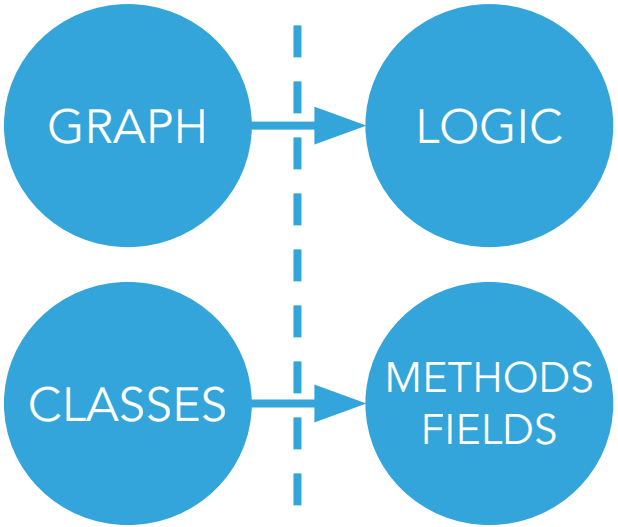
- 9 iterations is fast! ddmin uses 22 to reduce to an invalid program!
- We are faster because we model the validity of the program using a graph.
- Each closure is a valid program: our final program contains two.
- Scales: O(#final-closures * log #classes) vs ddmin O(#classes^2)

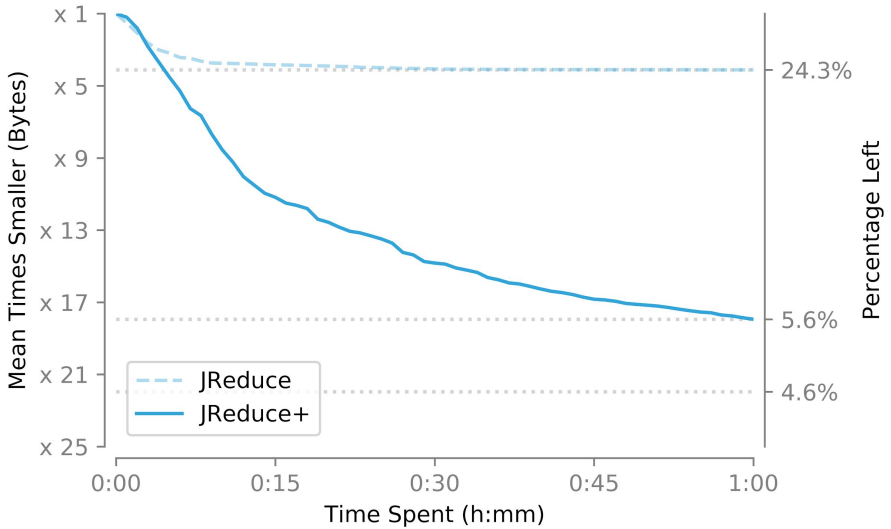


For more details, refer to Section 5.2 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#52-Run-JReduce-on-a-Simple-Example

UCLA

3. JREDUCE+: LOGICAL INPUT REDUCTION





INTEGRATED EVALUATION OF 3 TOOLS

- JReduce achieves 12x speed up on 100 large java projects in the NJR repository, compared to delta debugging [FSE 2019]
- JShrink achieves reduction of up to 47% and on average 14% on 22 maven-based applications from Github using BigQuery API. [FSE 2020]
- Accounting for dynamic language features in JShrink is indeed crucial to ensure behavior preservation for debloated software—reducing 98% of test failures incurred by a purely static equivalent, Jax, and 84% for ProGuard [FSE 2020]

Maven Wrapper Results

- Switch back and show that jinline -> jshrink -> jreduce ran

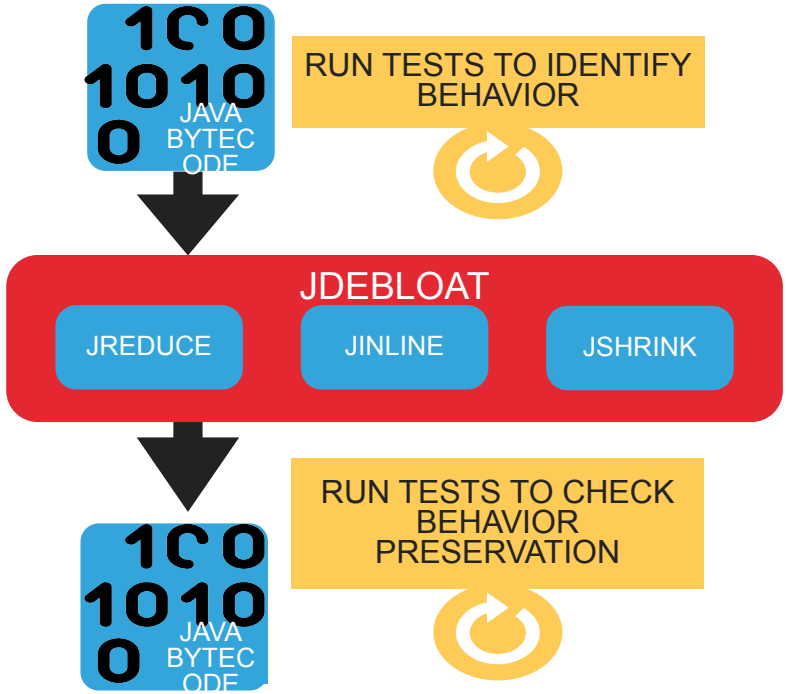
UCLA

SYNERGY EVALUATION

Order	Mean Size Reduction (%)	Median Size Reduction (%)
jreduce	40.2%	30.4%
jinline	14.6%	18.1%
jshrink	11.4%	3.1%
jinline+jshrink+jreduce	49.9%	30.3%

CONCLUSIONS

- We have developed three Java byte-code transformation tools with the goal to de-bloat and de-layer.
- JReduce, JShrink, and JInline *integrated together* can achieve 49.9% size reduction
- Lessons learned: Accounting for type safety and dynamic call *dependences* in a clever manner is important for test behavior preservation.
- Supporting *modern Java features* using dynamic profiling required significant engineering work.



PUBLICATIONS (1)

Christian Gram Kalhauge and Jens Palsberg, Binary Reduction of Dependency Graphs, ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE 2019.

Bobby Bruce, Tianyi Zhang, Jaspreet Arora, Guoqing Harry Xu, Miryung Kim, JShrink: In-depth Investigation into Debloating Modern Java Applications, ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE '20

Konner Macias, Mihir Mathur, Bobby R. Bruce, Tianyi Zhang, Miryung Kim, WebJShrink: A Web Service for Debloating Java Bytecode

PUBLICATIONS (2)

Gerenuk: Thin Computation over Big Native Data using Speculative Program Transformation, SOSP'19

Niijima: Sound and Automated Computation Consolidation for Efficient Multilingual Data-Parallel Pipelines, SOSP'19

PerfDebug: Performance Debugging of Computation Skew in Dataflow Systems, SoCC'19

Grapple: A Graph System for Static Finite-State Property Checking of Large-Scale Systems Code, EuroSys'19

Panthera: Holistic Memory Management for Big Data Processing over Hybrid Memories, PLDI'19

An Empirical Study of Common Challenges in Developing Deep Learning Applications, ISSRE'19

PUBLICATIONS (3)

White-Box Testing of Big Data Analytics with Complex User-Defined Functions, ESEC/FSE '19

A Formalization of Java's Concurrent Access Modes, OOPSLA'19.

What is Decidable about Gradual Types? POPL'20.

Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study, CHI'20

HeteroRefactor: Refactoring for Heterogeneous Computing with FPGA, ICSE'20

Low-Overhead Deadlock Prediction, ICSE'20.

Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks? ESEC/FSE '20

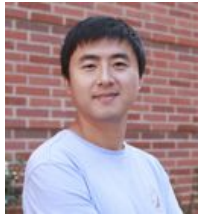
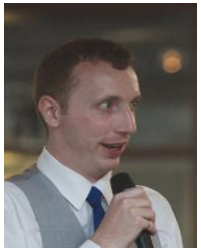
OPEN SOURCE & PUBLIC ARTIFACTS

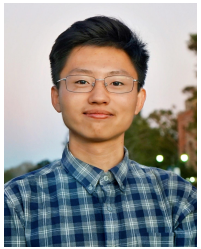
- JDebloat Download and Manual:
- http://debloating.cs.ucla.edu/debloat/index.html
- JShrink code, manual and datasets: https://doi.org/10.6084/m9.figshare.12435542 (FSE 2020 artefact badge)
- JReduce code, manual and datasets: https://dl.acm.org/do/10.5281/zenodo.3262201/full/ (FSE 2019 artifact badge)
- Internal Evaluation by CMU / SEI: https://github.com/tpcp-project/jdebloat

UCLA

JDEBLOAT:
JAVA BYTECODE DE-BLOATING AND DE-LAYERING




HANDS-ON TRIAL: MEDIUM
"MAVENWRAPPER" USING JINLINE

- Show the code before
- Add a call graph diagram. Constrast dynamic and static call
- After static inlining results: show the byte code and decompiled code

For more details, refer to Section 4.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#41-Run-JInline-on-Maven-wrapper

HANDS-ON TRIAL: HARD
"JUNIT" USING JREDUCE

- Create a hello world program with call dependency, run code
- Show the code before / Run Tests
- After delta debugging results: show the byte code and decompiled code
- Show the size reduction in numbers
- Describe your evaluation results from FSE 2019 in aggregate

UCLA

1 JSHRINK (A): UNUSED METHOD REMOVAL

```
class A{
  public A(){ }

  public String method_1(){
    return "A_String";
  }
}

class B extends A{

  public String foo = "foo";
  public String bar = "bar";

  public static void main(String[] args){
    B b = new B();

    System.out.println(b.method_1());
    System.out.println(b.method_2());
  }

  public B(){
    super();
  }

  public String method_2(){
    return this.foo;
  }

  public String method_3(){
    return this.bar;
  }
}
```

UCLA

1 JSHRINK (A): UNUSED METHOD REMOVAL

```
class A{
  public A(){ }

  public String method_1(){
    return "A_String";
  }
}

class B extends A{

  public String foo = "foo";
  public String bar = "bar";

  public static void main(String[] args){
    B b = new B();

    System.out.println(b.method_1());
    System.out.println(b.method_2());
  }

  public B(){
    super();
  }

  public String method_2(){
    return this.foo;
  }

  public String method_3(){
    return this.bar;
  }
}
```

Entry Point

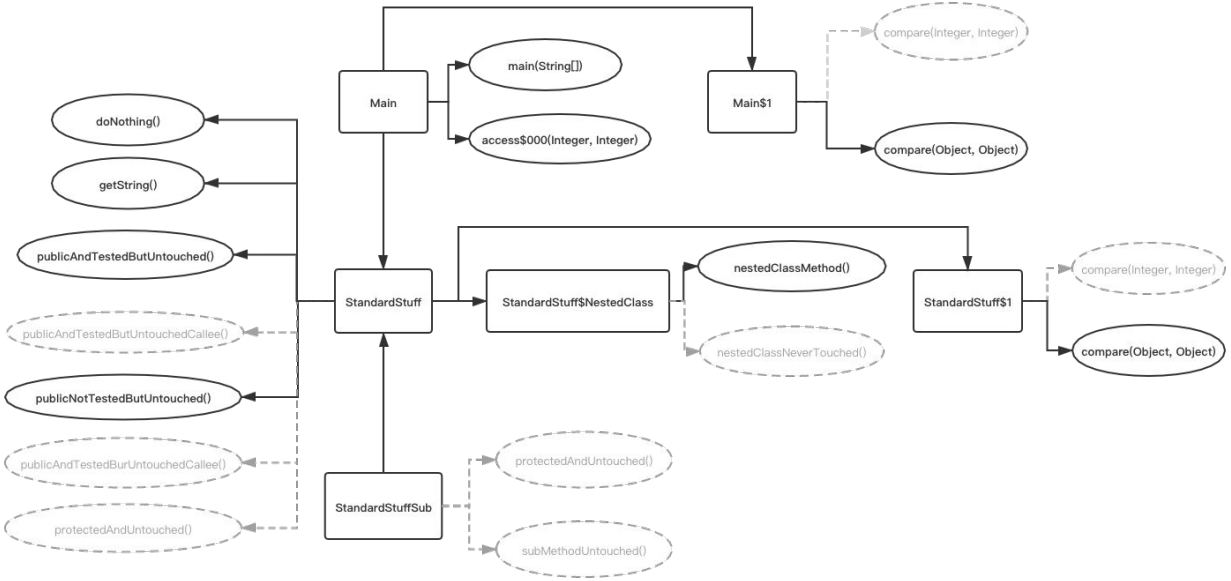
UCLA

1 JSHRINK (A): UNUSED METHOD REMOVAL

```
class A{
  public A(){ }

  public String method_1(){
    return "A_String";
  }
}

class B extends A{

  public String foo = "foo";
  public String bar = "bar";

  public static void main(String[] args){
    B b = new B();

    System.out.println(b.method_1());
    System.out.println(b.method_2());
  }

  public B(){
    super();
  }

  public String method_2(){
    return this.foo;
  }

  public String method_3(){
    return this.bar;
  }
}
```

UCLA

HANDS-ON TRIAL: EASY
"HELLOWORLD" USING JSHRINK



For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example

UCLA

HANDS-ON TRIAL: MEDIUM "MAVENWRAPPER" USING JSHRINK

- Run the following command:
  - ./jdebloat.py run jshrink --csv=tutorial --benchmark=ssss
- The JDebloat framework will only run JShrink and print out the debloating result as follows:

```
========================================================================
                        Debloating Stats:
========================================================================
Using debloating tool(s): jshrink
Benchmark ssss:
  Size before debloating: 91499
  Size after  debloating: 91499
  Size reduction: 0.00%
  Total test cases before debloating: 19
  Total test cases after  debloating: 19, (19 successes, 0 failures)
```

For more details, refer to Section 3.1 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#31-Run-JShrink-on-Maven-wrapper

TECHNICAL DETAIL EVALUATION RESULT

- While we are waiting the debloating results (~15 minute), we will discuss the results on security attack surface removal in the next slides.

REDUCE SECURITY ATTACK SURFACES

- De-serialization attacks craft a payload of serialized Java classes (i.e., gadgets), which executes arbitrary commands during deserialization.
- Ysoserial creates payloads to exploit 31 gadget chains in JDK and popular Java libraries. Ysoserial: https://github.com/frohoff/ysoserial
- Running our gadget-chain analysis, we detect two gadget chains in "dubbokeeper". Both gadget chains involve unsafe classes and methods in imported libraries from Spring Framework. **All gadget chains are removed after JShrink's de-bloating.**
- Comparison: Jax and JRed both removed the gadget chain but ProGuard removed only one of the two.

JShrink Basics

- Two key concepts
  - Call Graph
  - Java Reflection
- Let's look at some simple examples.

For more details, refer to Section 3.3 in our tutorial: http://debloating.cs.ucla.edu/debloat/SSSS20.html#33-Run-JShrink-on-a-Simple-Example