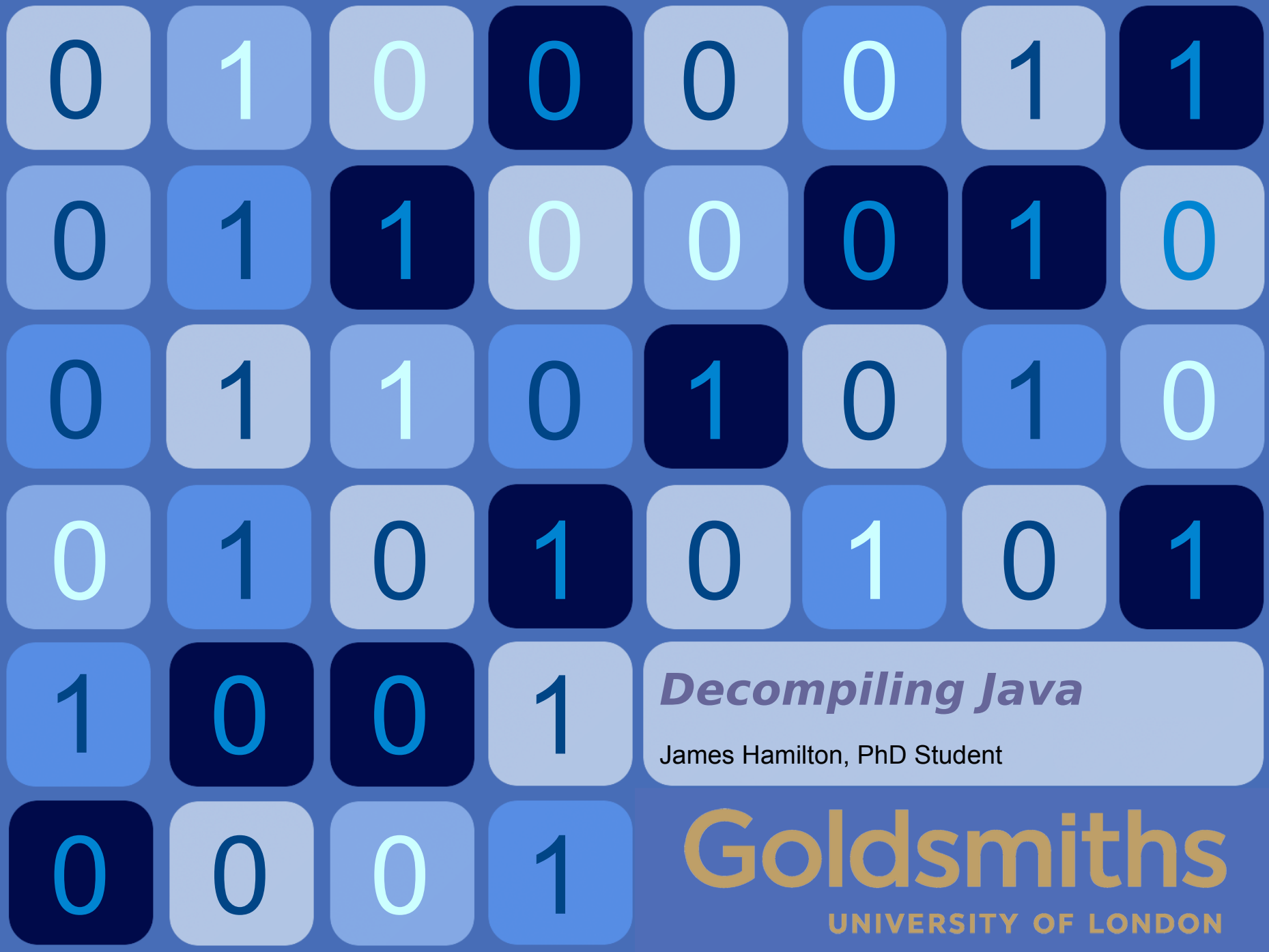

0 1 0 0 0 0 1 1
0 1 1 0 0 0 1 0
0 1 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 0 1
0 0 0 1
Decompiling Java
James Hamilton, PhD Student
Goldsmiths
UNIVERSITY OF LONDON

What we did

- We tested 10 different decompilers with the latests Java class file format using 10 different test programs.

decompiler	type	status
Mocha	free	obsolete
SourceTec	commercial	obsolete
SourceAgain	commercial	obsolete
Jad	free	unmaintained
JODE	open-source	unmaintained
ClassCracker3	commercial	obsolete
jReversePro	open-source	unmaintained
Dava	open-source	current
jdec	open-source	current
Java Decompiler	free	current

- In 2003, Emmerik tested the first 8 + 2 other, unobtainable ones.

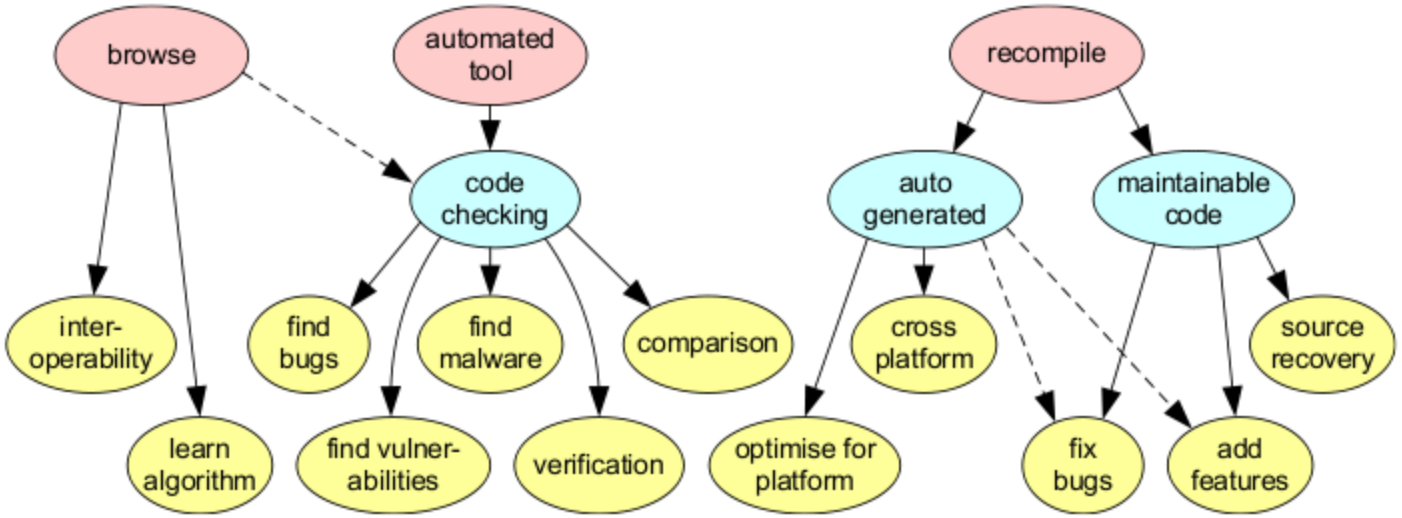
What is a Java Decompiler?

- transforms Java bytecode into Java source code

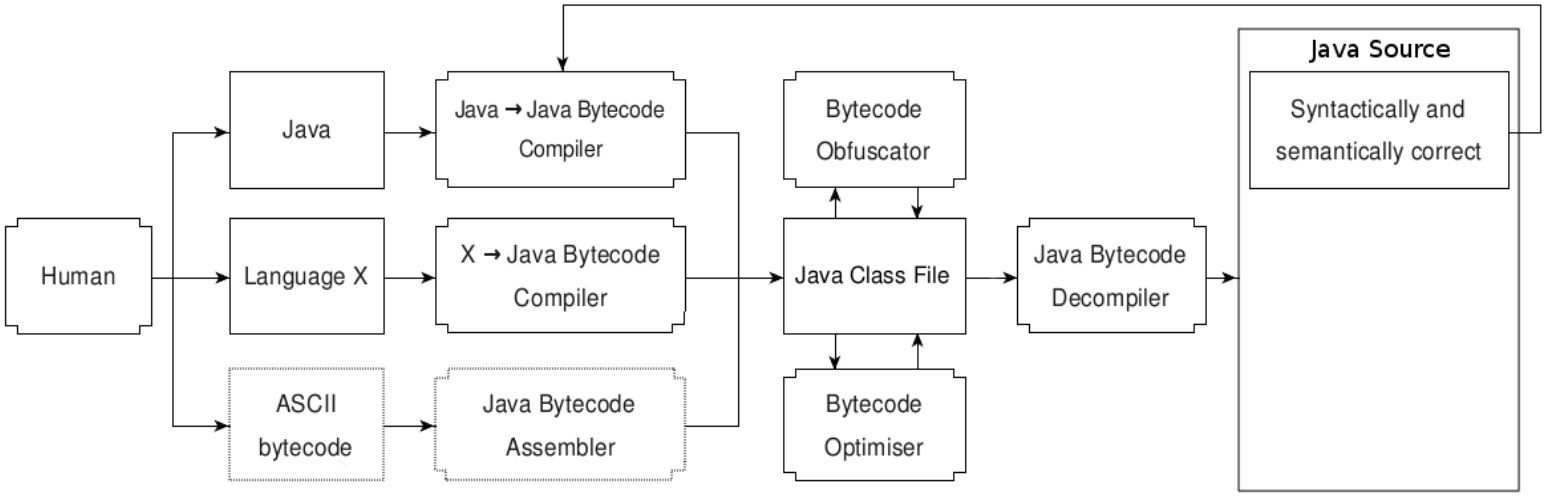
```
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0:    aload_0
   1:    invokespecial    #1; //Method java/lang/Object."<init>":()V
   4:    return

public static void main(java.lang.String[]);
  Code:
   0:    getstatic    #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:    ldc          #3; //String Hello World
   5:    invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   8:    return

}
```

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

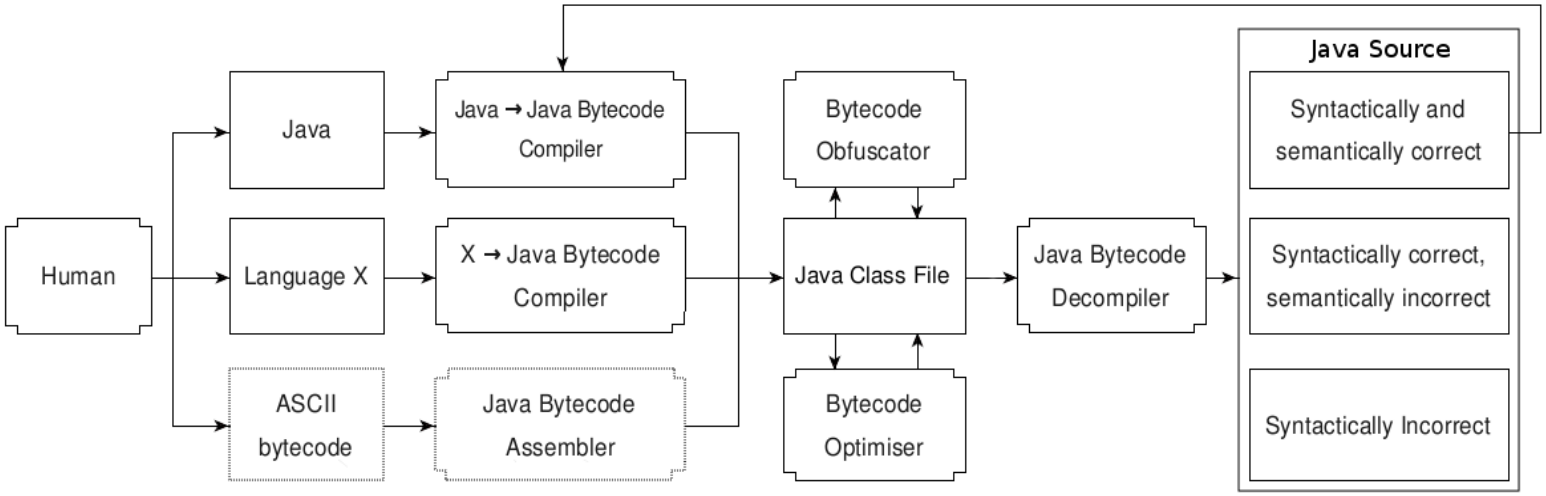
What are they for?



Java bytecode cycle



Java bytecode cycle

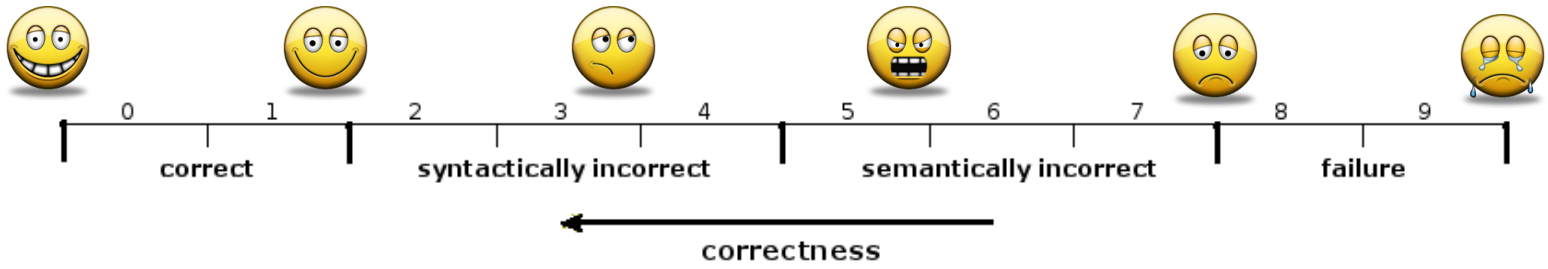


Assumption about how the bytecode was generated

- Java compiler generated bytecode
  - most likely Sun's *javac*
- aribtrary bytecode
  - generated by another language to Java bytecode compiler
  - generated manually by a programmer
  - transformed Java compiled bytecode

Our evaluation

- 10 decompilers tested
- 10 programs testing different problem areas
  - 6 *javac* generated
  - 4 abitrary
- based on a 2003 survey
- effectiveness scale

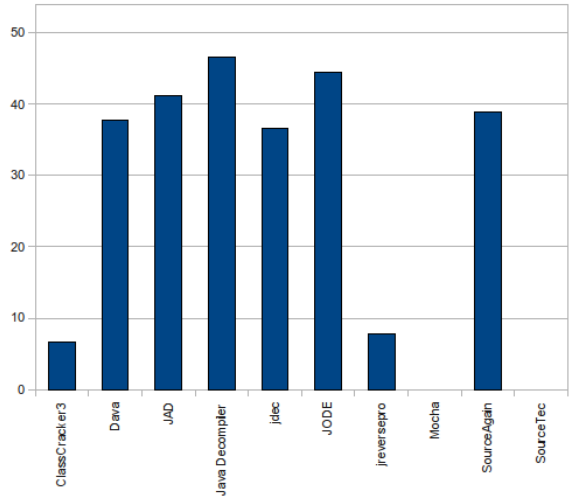


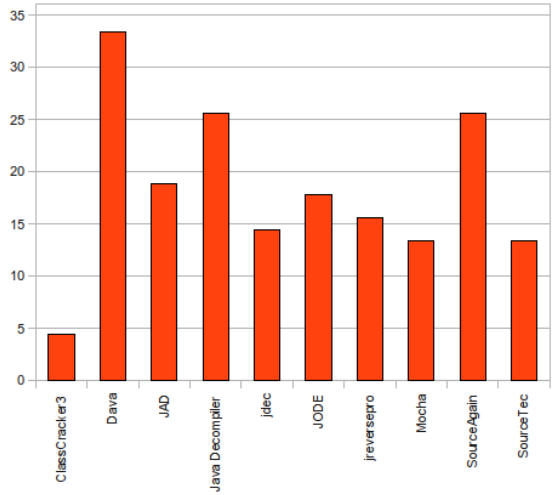
- \+ indication of the general effectiveness of a Java decompiler.
- \- manual inspection and subjective scale.

Emmerik, 2003, http://www.program-transformation.org/Transform/JavaDecompilerTests

Results

- many of the current decompilers cannot parse the latest class files
  - 4 obsolete
  - 3 unmaintained
- no currently maintained commercial decompilers
- best decompiler? depends on the tool which generated the class file
  - Dava for *abitrary* bytecode
  - Java Decompiler or JODE for *javac* bytecode





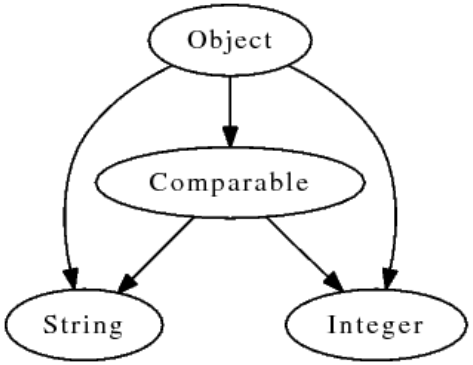
The sorts of problems the Java decompilers had

- Type inference for local variables – giving the tighest possible type to local variables
- Exception handling
- Control flow

Where decompilers failed

Type Inference

```
public class TypeSet {
   public void m1(Integer i, String s) {
            untyped x;

            if(i != null)
                     x = i;
            else
                     x = s;
            m2(x);      invokevirtual TypeSet/m2(Ljava/lang/Comparable;)V
   }

   public void m2(Comparable x) { }
}
```



• this example is easy but multiple inheritance via interfaces complicates things*.

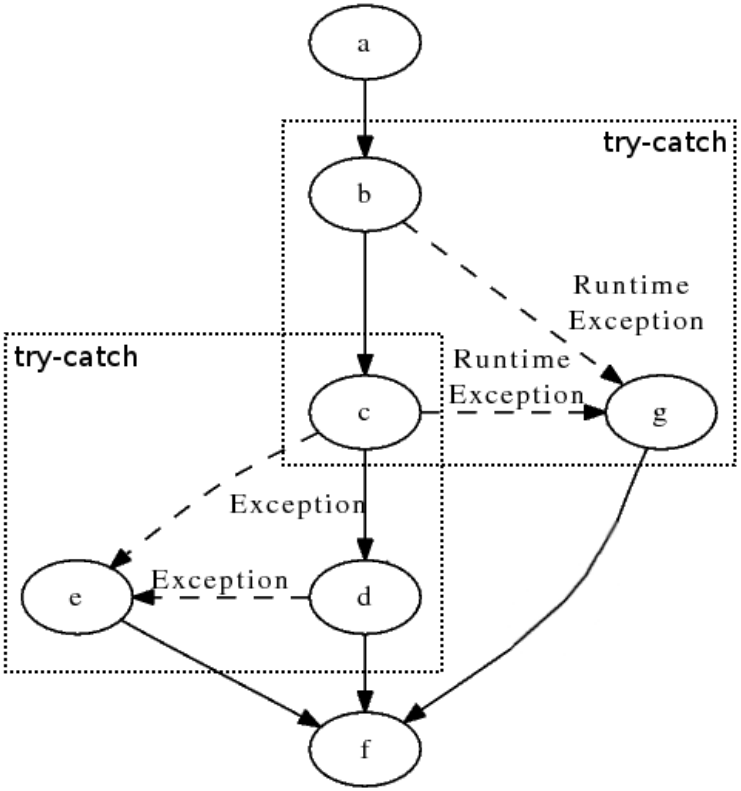
• is this really a problem, if all we need is compilable code?
• can we just make everything an object, and use typecasts?

Example from: Java and the Java Virtual Machine: Definition, Verification and Validation, Stärk et al, 2001

* see Efficient Inference of Static Types for Java Bytecode, Gagnon et al. 2000 and Efficient local type inference, Bellamy et al. 2008.

Where decompilers failed

Intersecting try-catch blocks




```
System.out.println("a");

label_0: {
    try {
        System.out.println("b");
     }catch (RuntimeException $r9) {
        System.out.println("g");
        break label_0;
     }
     try {
        System.out.println("c");
      }catch (RuntimeException $r9) {
        System.out.println("g");
        break label_0;
      }catch (Exception $r6) {
        System.out.println("e");
        break label_0;
      }
      try {
        System.out.println("d");
       }catch (Exception $r6) {
         System.out.println("e");
       }
} //end label_0:

System.out.println("f");
```

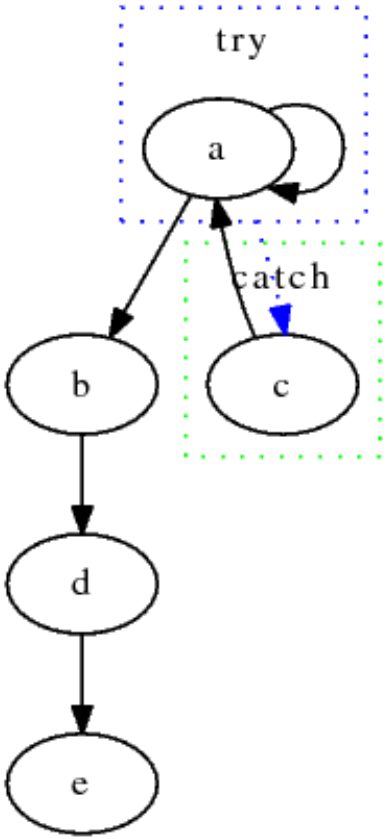
most decompilers, except Dava, do not expect this.

Example from: Decompiling Java Bytecode: Problems, Traps and Pitfalls, Miecznikowski et al, 2002

Where decompilers failed

Control Flow

```
public static int foo(int i, int j) {
        while (true) {
            try{
                while (i < j)
                    i = j++/i;

            }catch (RuntimeException re) {
                i = 10;
                continue;
            }
            break;
        }
        return j;
}
```



Example from: Decompiling Java Bytecode: Problems, Traps and Pitfalls, Miecznikowski et al, 2002

Conclusion

- Not much improvement in 6 years
  - all decompilers aren't very good
- Problems caused by newer class file specification
- Different decompilers for different problems

Questions

- can we produce a decompiler which combines the best of *javac* decompilers and *abitrary* decompilers?
- do we really need to perform type inference if the only condition is that we require re-compilable code?
- are the problems with the Java decompilers fundamental or just bugs?