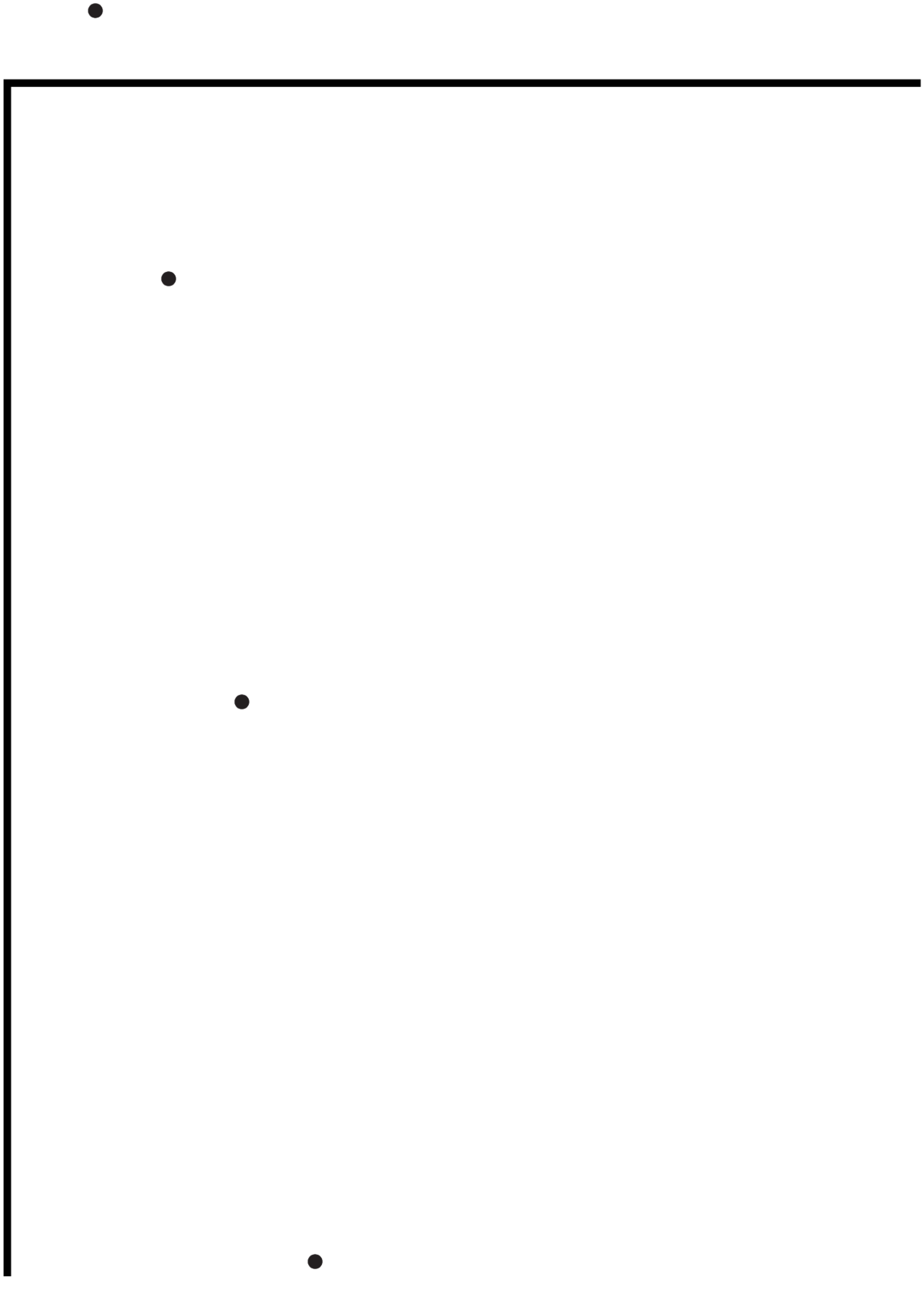
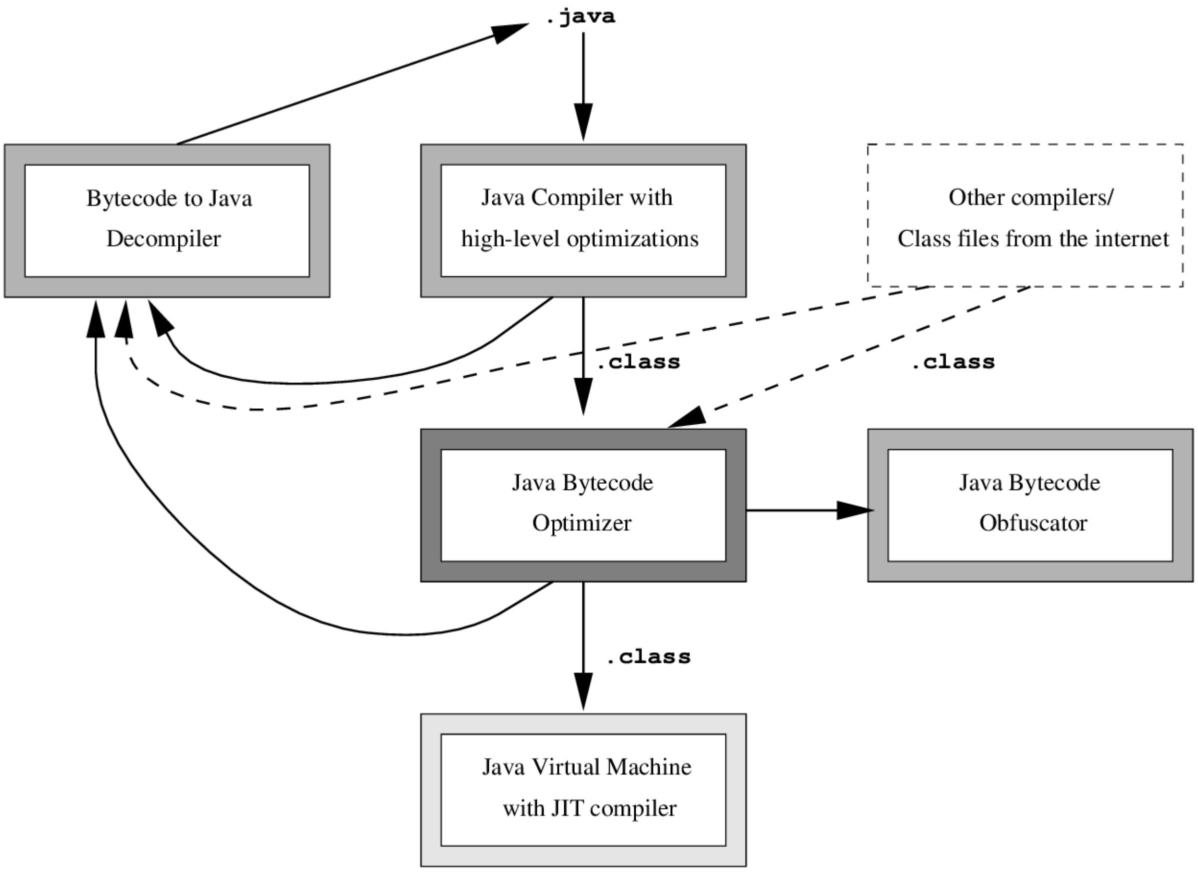
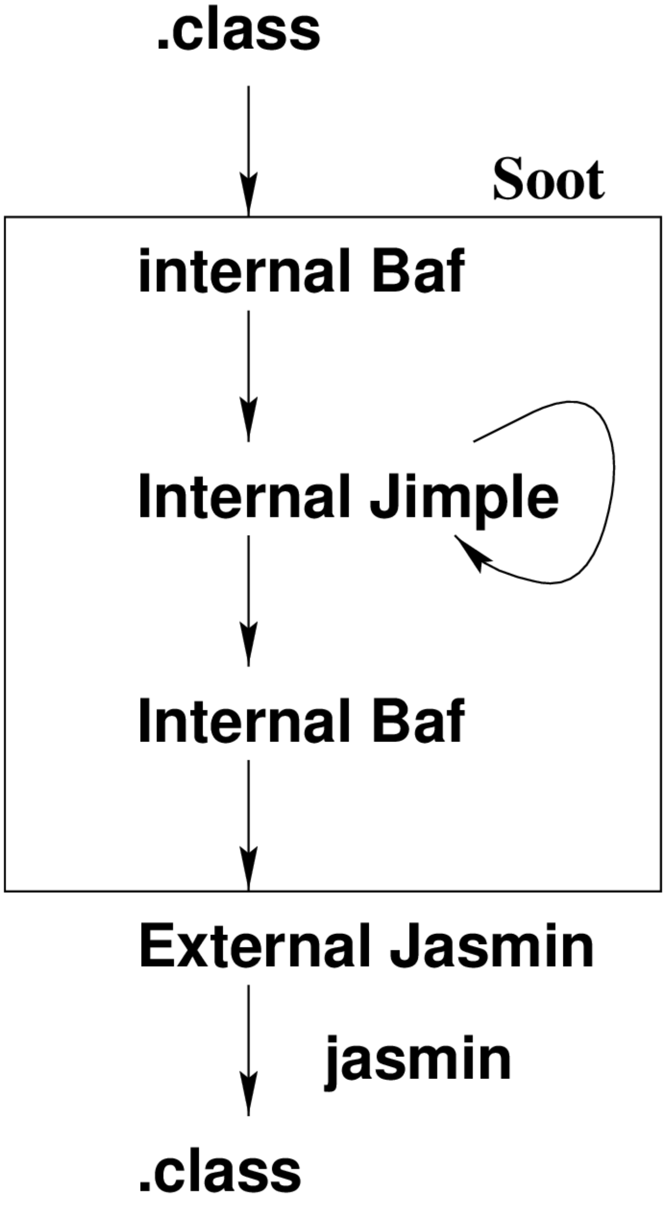



.java
Bytecode to Java Decompiler
Java Compiler with high-level optimizations
Other compilers/ Class files from the internet
.class
.class
Java Bytecode Optimizer
Java Bytecode Obfuscator
.class
Java Virtual Machine with JIT compiler


Specification
Sable Compiler Compiler (SableCC)
java code

→

→



.class

Soot

internal Baf

Internal Jimple

Internal Baf

External Jasmin

jasmin

.class

```
import ca.mcgill.sable.soot.*;
import ca.mcgill.sable.soot.jimple.*;
import java.io.*;

// Prints the given class in .jimple format.
public class PrintClass
{ public static void main(String[] args)
  { // Retrieve and print class.
    SootClassManager cm = new SootClassManager();
    SootClass sClass = cm.getClass(args[0]);
    PrintWriter out =
                  new PrintWriter(System.out, true);

    sClass.printTo(new BuildBody(Jimple.v(),
                     new StoredBody(ClassFile.v())),
                   out);
  }
}
```

```
StmtList stmtList = body.getStmtList();
ListIterator stmtIt = stmtList.listIterator();

while(stmtIt.hasNext())
{ Stmt s = (Stmt) stmtIt.next();
  if (s instanceof GotoStmt)
  { // Back up before the GotoStmt
    stmtIt.previous();
    // insert "tmpLocal = gotoCounter;"
    stmtIt.add(Jimple.v().newAssignStmt(tmpLocal,
       Jimple.v().newStaticFieldRef(gotoCounter)));
    // skip over statement just added
    stmtIt.next();
    // insert "gotoCounter = tmpLocal + 1L;"
    stmtIt.add(Jimple.v().newAssignStmt(
      Jimple.v().newStaticFieldRef(gotoCounter),
      Jimple.v().newAddExpr(tmpLocal,
                             LongConstant.v(1L))));
    // skip over statement just added
    stmtIt.next();
    // skip over the goto
    stmtIt.next();
  }
}
```

```
public int stepPoly(int x)
{ if (x < 0)
    { System.out.println("error"); return -1; }
  else if (x <= 5)
    return x * x;
  else
    return x * 5 + 16;
}
```

```
public int stepPoly(int)
{ java.io.PrintStream r1;
  Example r0;
  int i0, i1, i2, i3;
  r0 := @this;
  i0 := @parameter0;
  if i0 >= 0 goto label0;
  r1 = java.lang.System.out;
  r1.println("error");
  return -1;
label0:
  if i0 > 5 goto label1;
  i1 = i0 * i0;
  return i1;
label1:
  i3 = i0 * 5;
  i2 = i3 + 16;
  return i2;
}
```

direRted edges represent sRRRlass

softnodes are RiRple RariaRlesR type needs

- to Re deRned

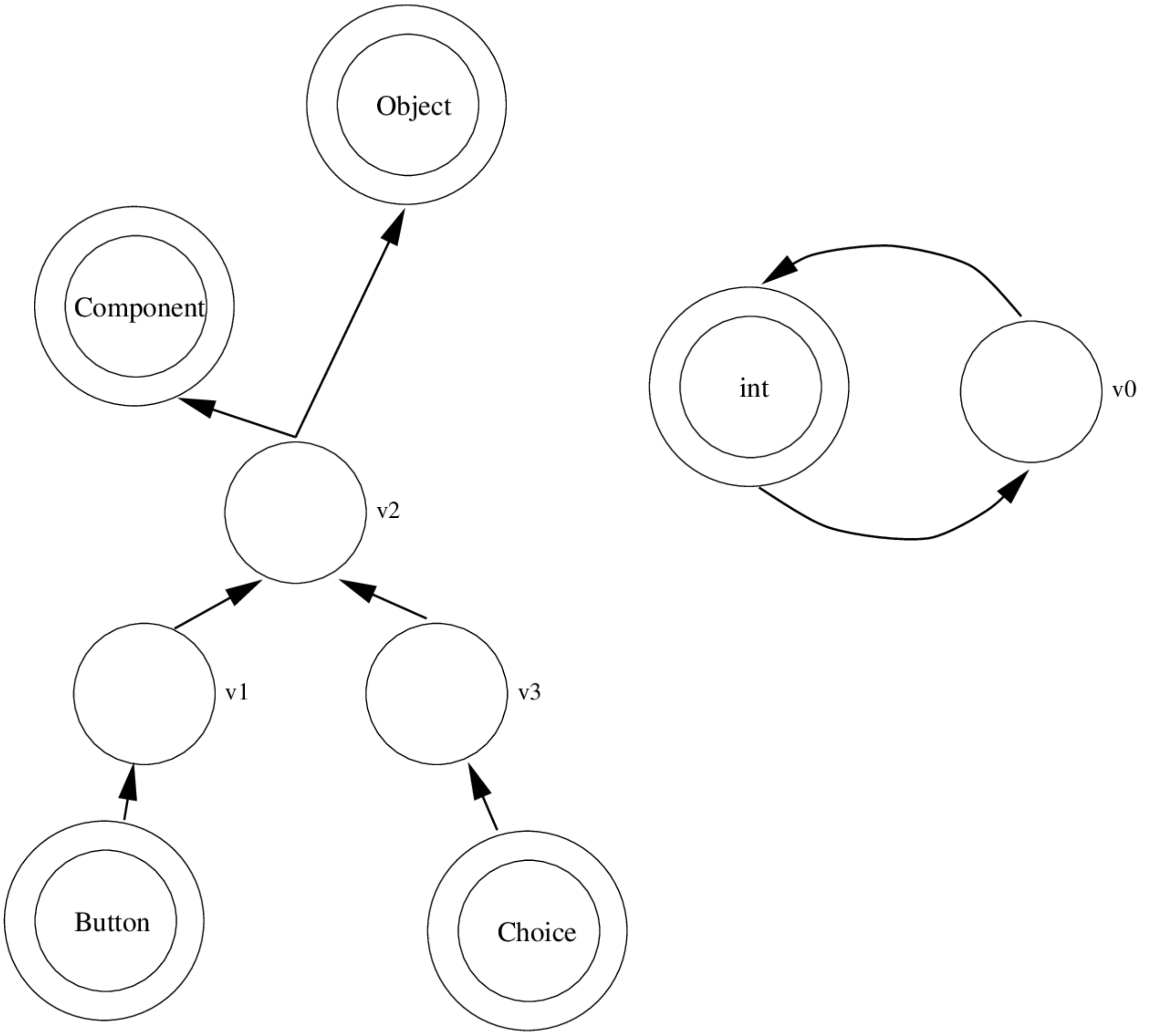
hard nodes are types in the type hierarRhy

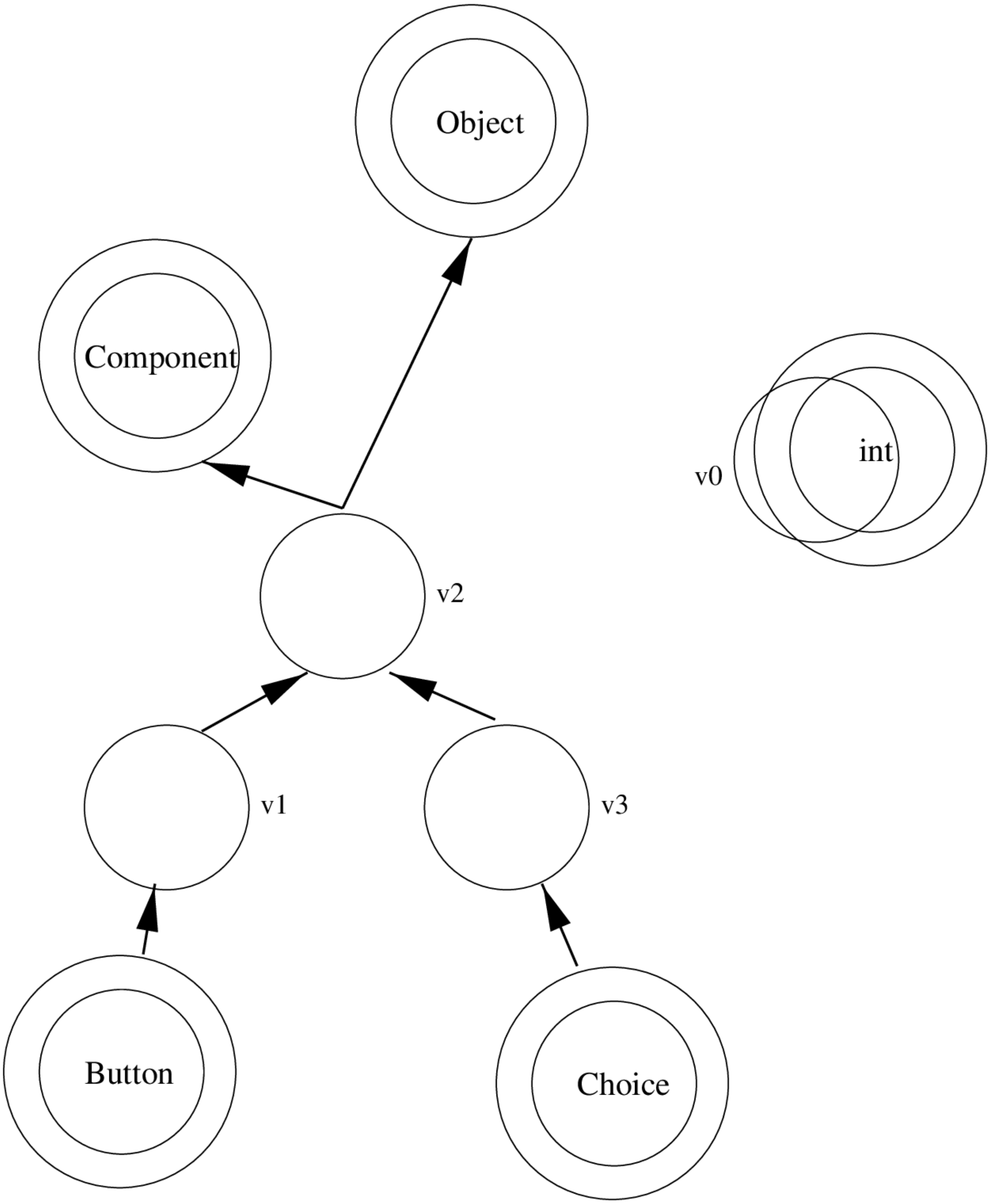
relationships iRposed Ry RiRple stateRents

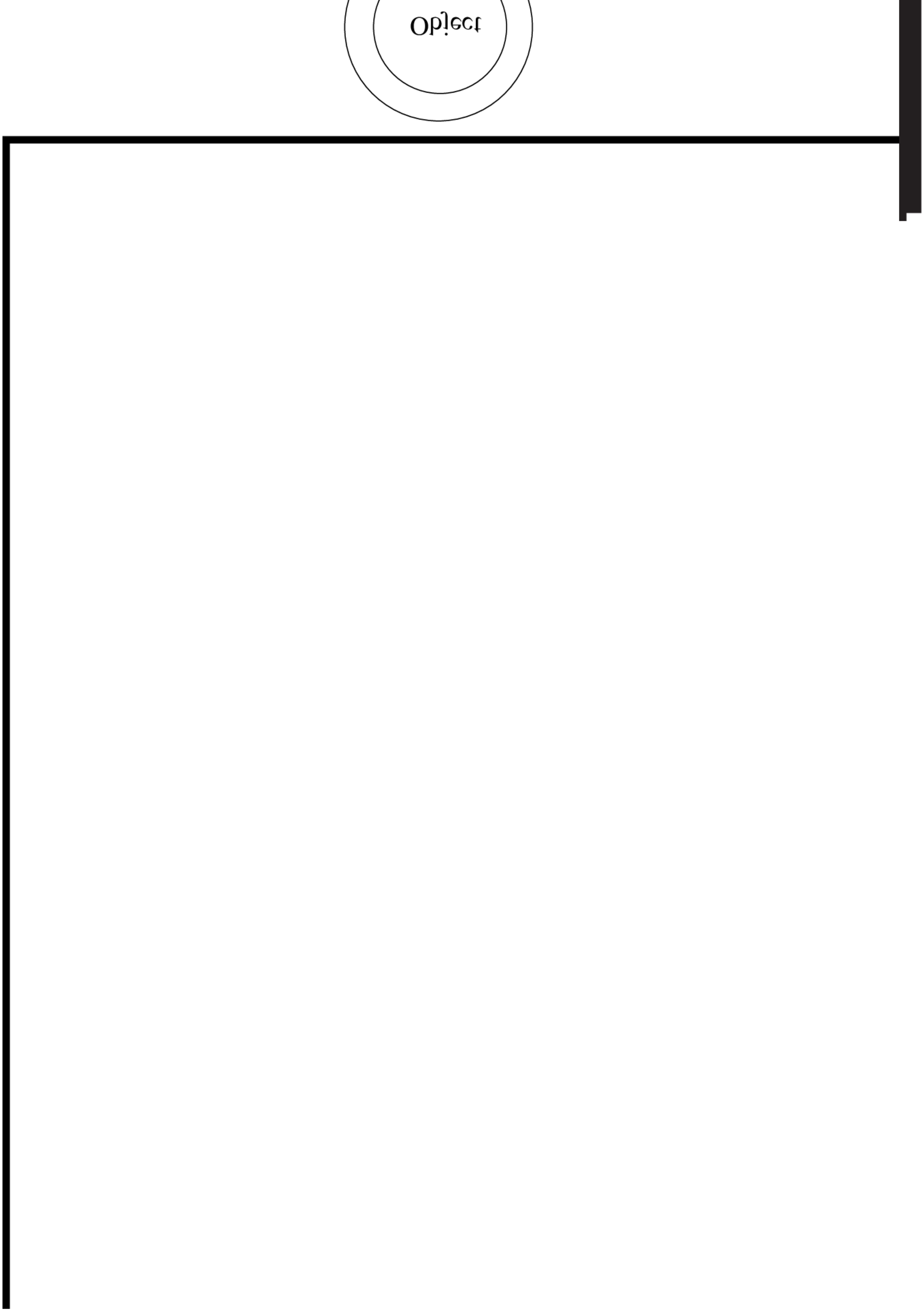
-

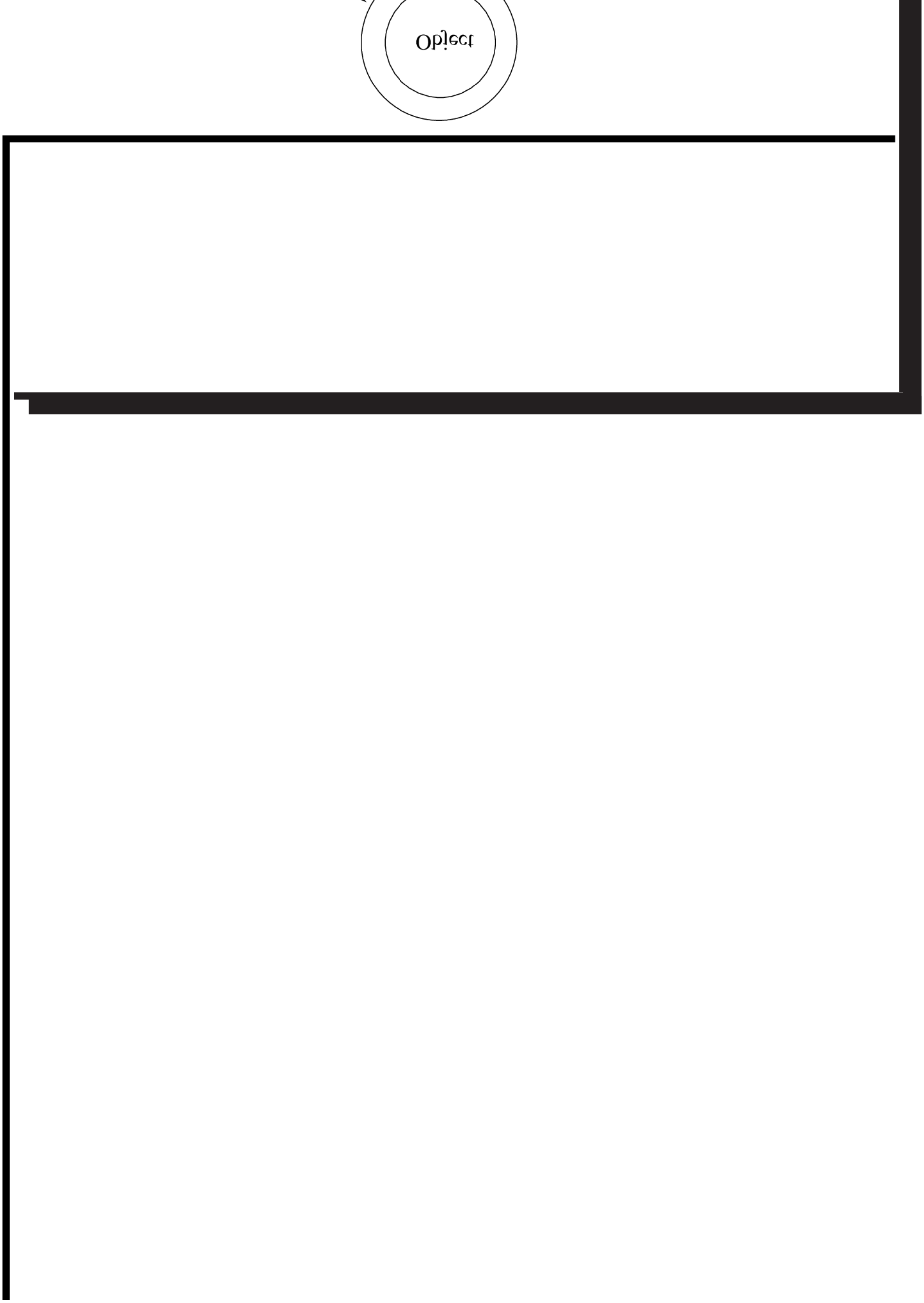
- hard node is Rollapsed Rith a soft

```
import java.awt.*;

class Test
{ static Component method(boolean b)
  { Component c = new Button();
    if(b) c = new Choice();
    Object o = c;
    o.notify();
    return c;
  }
}

class Test extends java.lang.Object
{ static java.awt.Component method(boolean param0)
  { unknown_type v0;
    unknown_type v1;
    unknown_type v2;
    unknown_type v3;
    v0 := param0;
    v1 = new java.awt.Button;
    specialinvoke v1.[void java.awt.Button.<init>()]();
    v2 = v1;
    if v0 == 0 goto label0;
    v3 = new java.awt.Choice;
    specialinvoke v3.[void java.awt.Choice.<init>()]();
    v2 = v3;
 label0:
    virtualinvoke v2.[void java.lang.Object.notify()]();
    return v2;
  }
}
```


Object
Component
v2
v1
v3
Button
Choice
int
v0


Object
Component
int
v0
v2
v1
v3
Button
Choice

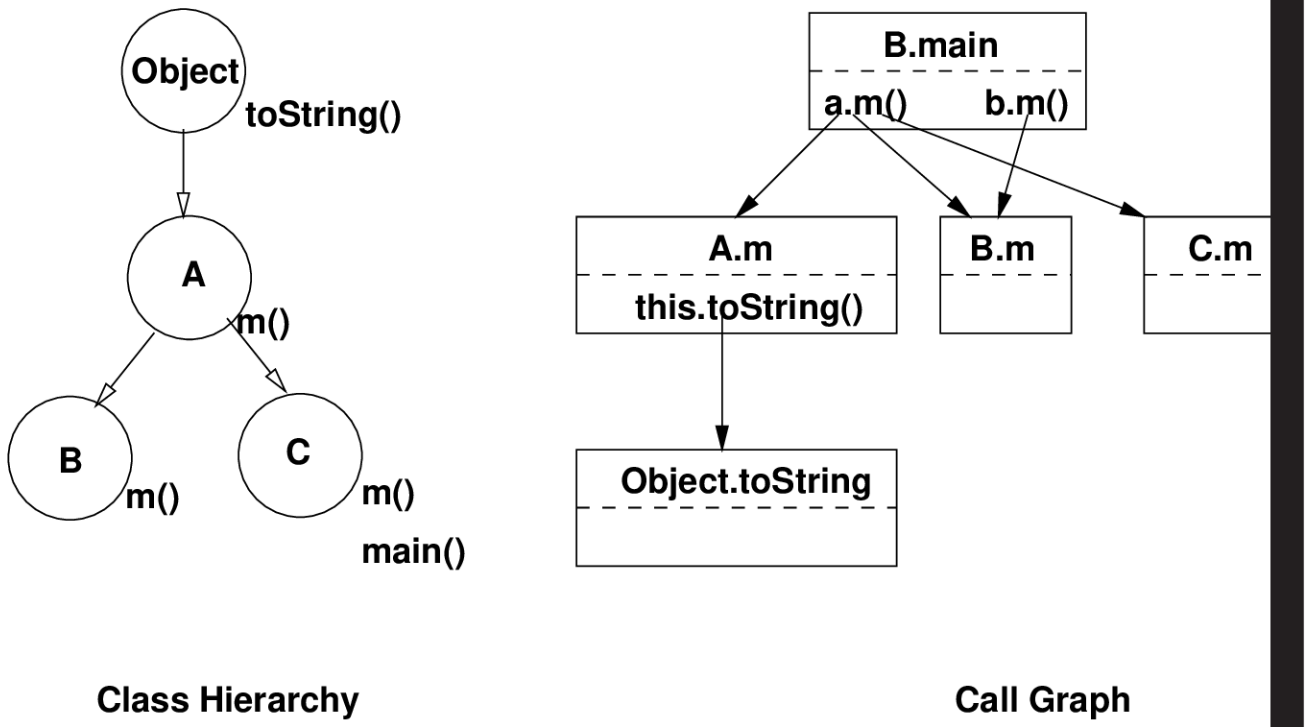

Object


Object

```
import java.awt.*;

class Test
{ static Component method(boolean b)
```

lines per method (#)	methods (#)	avg. execution time (ms)

```
class A extends Object {
  String m() {
    return(this.toString());
  }
}

class B extends A {
  String m() { ... }
}

class C extends A {
  String m() { ... }
  public static void main(...) {
   A a = new A();
   B b = new B();
   String s;
   ...
   s = a.m();
   s = b.m();
  }
}
```



- *Rapid type analysis reRnes hierarRhy analysis Ry eRRlRding Ralls that are dRe to Rlasses that haRe no instanResR*

type propagation graphR • *RhereR*