Androlic: An Extensible Flow, Context, Object, Field, and Path-sensitive Static Analysis Framework for Android

Linjie Pan, Baoquan Cui, Jiwei Yan, Xutong Ma,
Jun Yan and Jian Zhang

Institute of Software, Chinese Academy of Sciences
Presenter: Linjie Pan

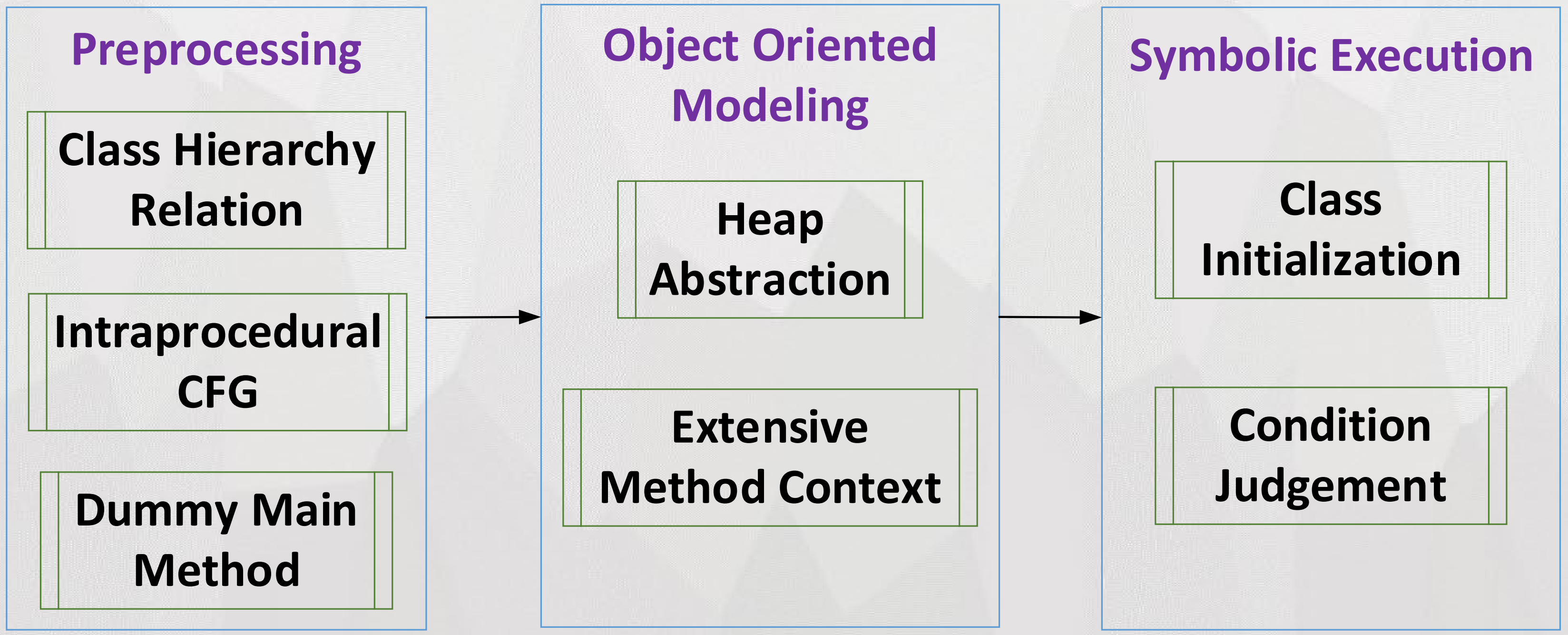
Introduction

- The precision of static analysis heavily depends on sensitivity (flow, context, path, object, field)
- Most Android static analysis tools only consider a few of the five sensitivities [1]
- Most analysis tools are difficult to extend because they are designed for concrete analysis tasks
- We propose Androlic which considers five sensitivities and is easy to extend

[1] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and Y. L. Traon. Static analysis of Android apps: A systematic literature review. Information & Software Technology, 88:67–95, 2017.

Androlic

- Preprocessing
- Object oriented modeling
- Symbolic execution



Heap Abstraction (Object and Field Sensitivity)

- Maintain a map from reference variables to allocation sites
- The map can only be built and updated through *AssignStmt*
- Right operands: two types of allocation sites
  - Explicit: *NewExpr* (*new A()*), *NewArrayExpr* (*new int[4]*)
  - Implicit: *InvokeExpr* of library method
- Left operands: four types of variables
  - ArrayRef (ar[1])
  - StaticFieldRef (A.sf)
  - InstanceFieldRef (b.if)
  - Local (c)
- Process the statements of entry methods along its CFG and update the map

Extensive Context (Context Sensitivity)

- Perform inter-procedural analysis when dealing with invocation of non-library methods
- Extensive context
  - Actual parameters of invoked methods (common context)
  - Base of non-static invoked methods (object)
  - Static fields that could appear within the invoked methods
- *v.methodName(parameterList), v* is the base
  - The type of v decides which method is invoked
  - Context variables can be operated within the invoked method
- Map context variables to the allocation sites they point to
- Build call graph on the fly

Condition Checking (Path Sensitivity)

- Check the feasibility of branches
- If all variables in *ConditionExpr* correspond to concrete values, we can get a definite result
    - Numeric constant
    - String constant
    - Null constant
    - Explicit allocation site
- Otherwise, all successive branches are taken as feasible branches
- Clone heap model for each branch

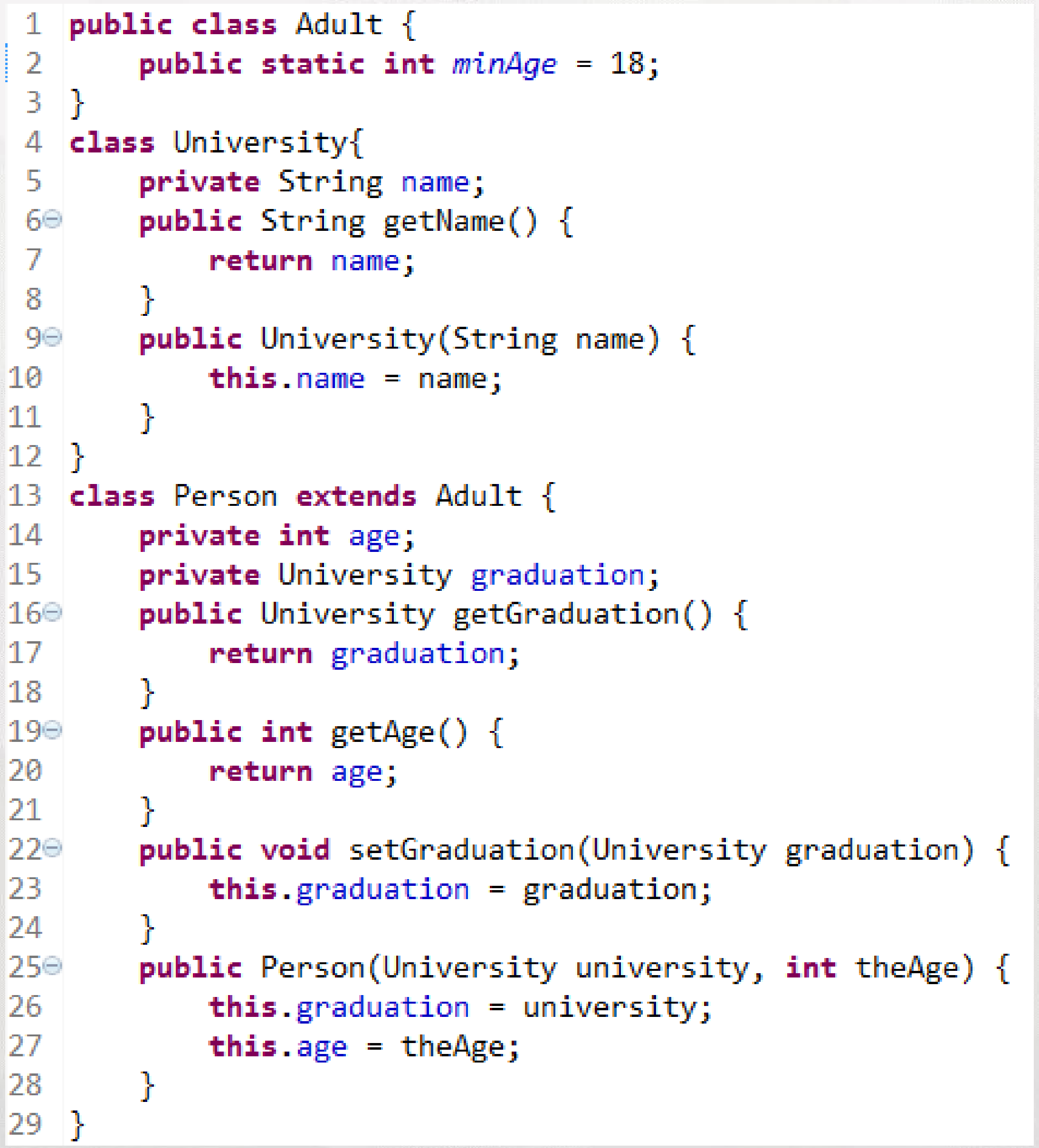
Implementation

- Androlic is built on top of Soot and Jimple (10000+ sloc)
- Latest version is put on github (https://github.com/pangeneral/Androlic)
- Configuration
  - MaxPathNum (40000)
  - MaxRecursiveNum (0)
  - MaxUnrollingNum (1)
  - MaxRunningTime (30 minutes)
  - EntryMethod (dummy main method)
  - …

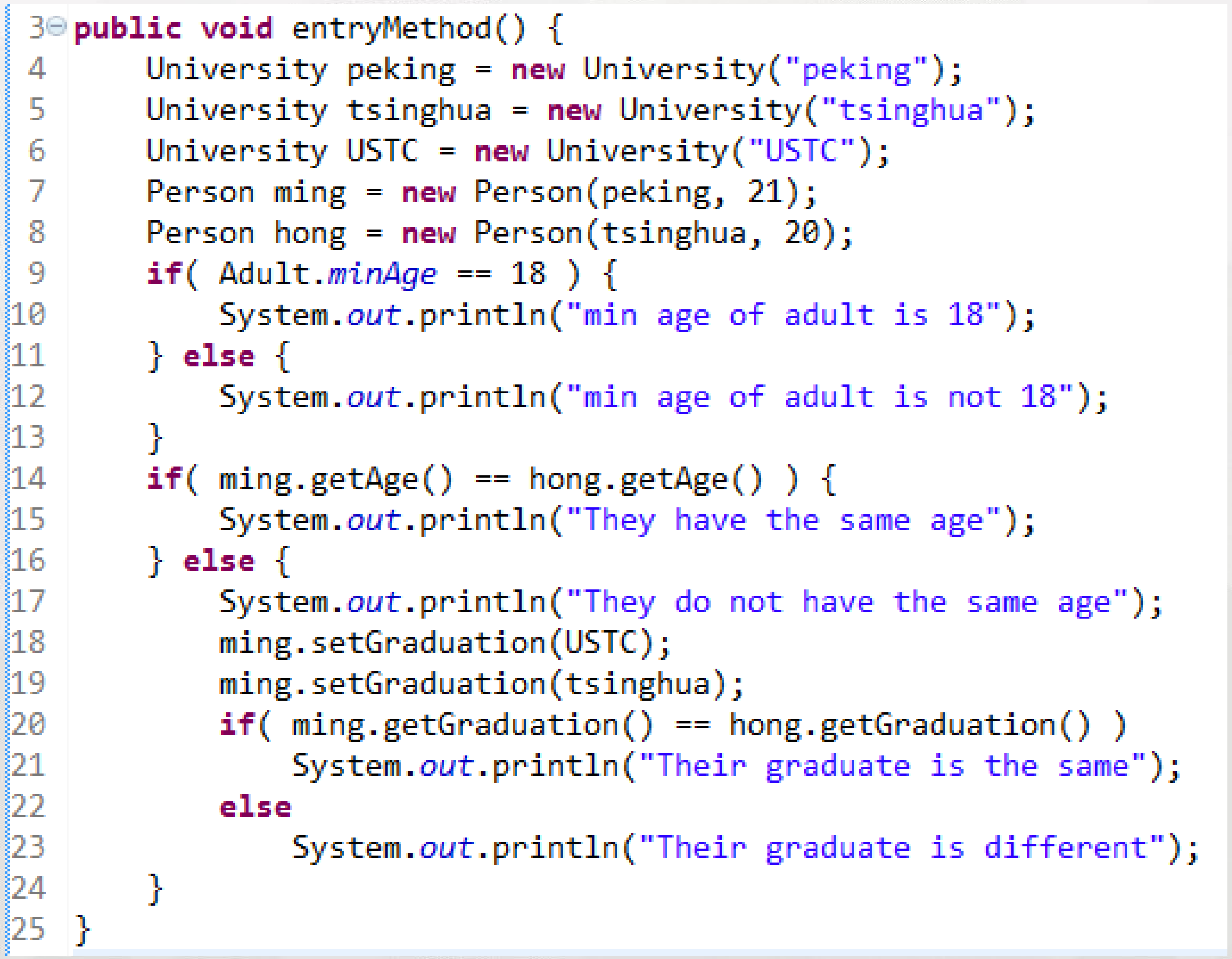
Extensibility

- ISymbolicEngineInstrumenter
  - public void onPreStmtExecution(Unit currentUnit, GlobalMessage globalMessage);
  - public void onPostStmtExecution(Unit currentUnit, GlobalMessage globalMessage);
  - public void onExceptionProcess(Unit currentUnit, GlobalMessage globalMessage, AbstractAndrolicException exception);
- ILibraryInvocationProcessor
  - public IBasicValue getLibraryInvocationReturnValue(AssignStmt stmt, InvokeExpr libraryInvokeExpr, ContextMessage context, GlobalMessage globalMessage);
  - public boolean processLibraryInvocation(InvokeStmt stmt, InvokeExpr libraryInvokeExpr, ContextMessage context, GlobalMessage globalMessage);
- INewExprProcessor
  - public NewRefHeapObject getNewHeapObject(AssignStmt stmt, NewExpr newExpr,ContextMessage context, GlobalMessage globalMessage);
- INewArrayExprProcessor
  - public NewArrayHeapObject getNewArrayHeapObject(AssignStmt stmt, NewArrayExpr newArrayExpr, ContextMessage context, GlobalMessage globalMessage);

Case Study

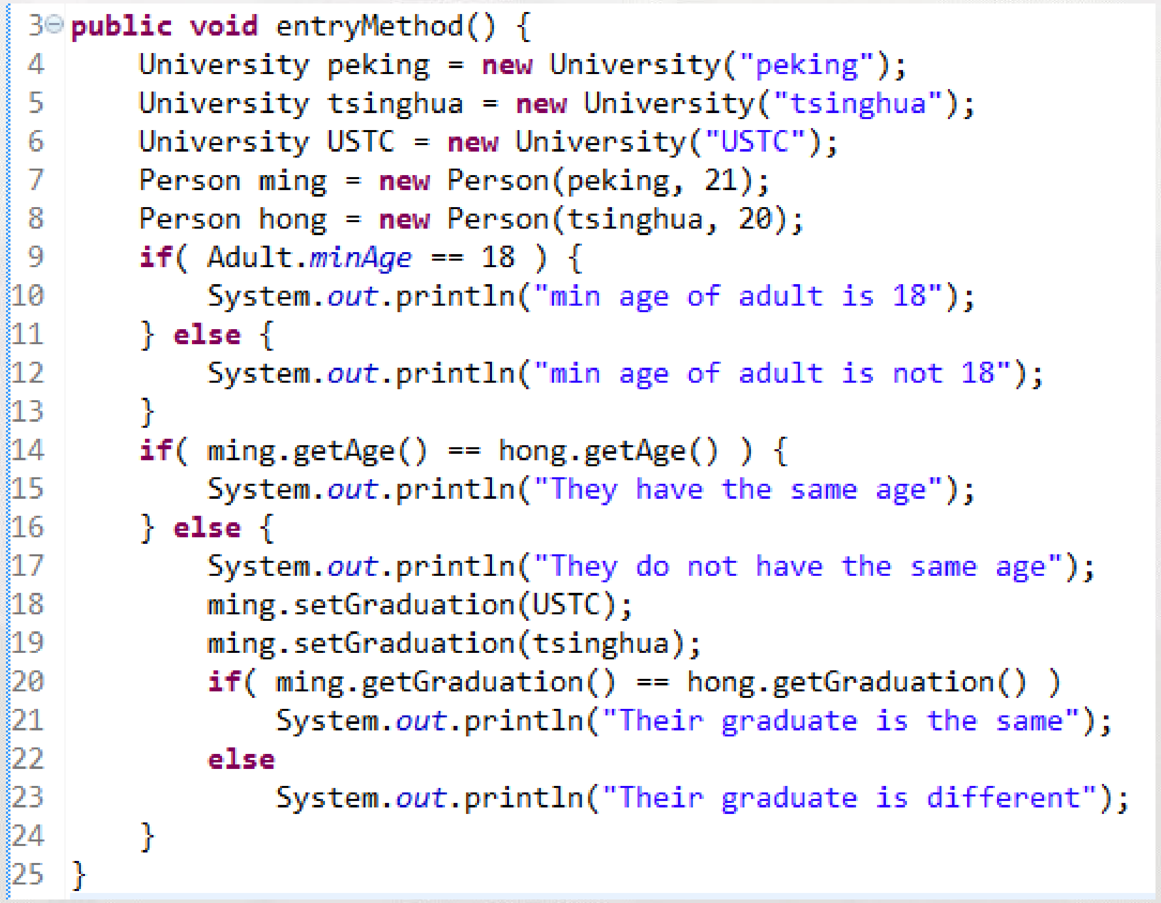


```java
public class Adult {
    public static int minAge = 18;
}
class University{
    private String name;
    public String getName() {
        return name;
    }
    public University(String name) {
        this.name = name;
    }
}
class Person extends Adult {
    private int age;
    private University graduation;
    public University getGraduation() {
        return graduation;
    }
    public int getAge() {
        return age;
    }
    public void setGraduation(University graduation) {
        this.graduation = graduation;
    }
    public Person(University university, int theAge) {
        this.graduation = university;
        this.age = theAge;
    }
}
```

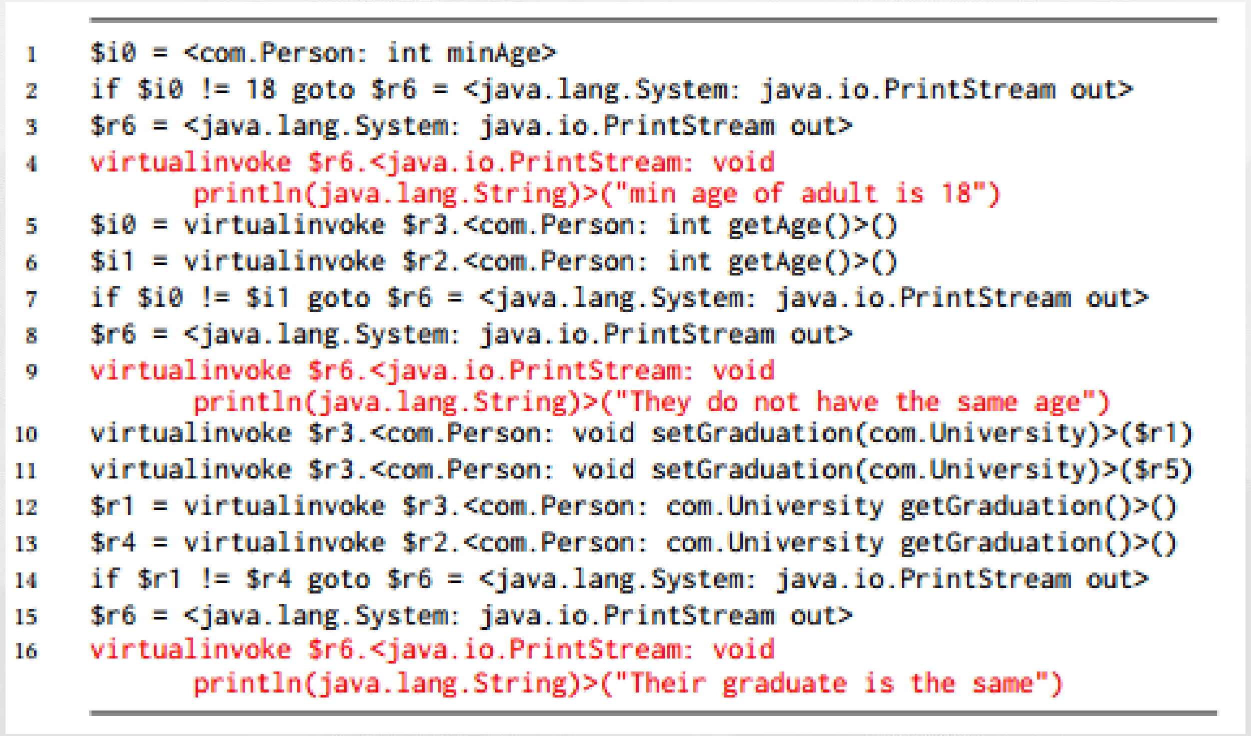


```java
public void entryMethod() {
    University peking = new University("peking");
    University tsinghua = new University("tsinghua");
    University USTC = new University("USTC");
    Person ming = new Person(peking, 21);
    Person hong = new Person(tsinghua, 20);
    if( Adult.minAge == 18 ) {
        System.out.println("min age of adult is 18");
    } else {
        System.out.println("min age of adult is not 18");
    }
    if( ming.getAge() == hong.getAge() ) {
        System.out.println("They have the same age");
    } else {
        System.out.println("They do not have the same age");
        ming.setGraduation(USTC);
        ming.setGraduation(tsinghua);
        if( ming.getGraduation() == hong.getGraduation() )
            System.out.println("Their graduate is the same");
        else
            System.out.println("Their graduate is different");
    }
}
```

Result of Case Study



```java
public void entryMethod() {
    University peking = new University("peking");
    University tsinghua = new University("tsinghua");
    University USTC = new University("USTC");
    Person ming = new Person(peking, 21);
    Person hong = new Person(tsinghua, 20);
    if( Adult.minAge == 18 ) {
        System.out.println("min age of adult is 18");
    } else {
        System.out.println("min age of adult is not 18");
    }
    if( ming.getAge() == hong.getAge() ) {
        System.out.println("They have the same age");
    } else {
        System.out.println("They do not have the same age");
        ming.setGraduation(USTC);
        ming.setGraduation(tsinghua);
        if( ming.getGraduation() == hong.getGraduation() )
            System.out.println("Their graduate is the same");
        else
            System.out.println("Their graduate is different");
    }
}
```



```
$i0 = <com.Person: int minAge>
if $i0 != 18 goto $r6 = <java.lang.System: java.io.PrintStream out>
$r6 = <java.lang.System: java.io.PrintStream out>
virtualinvoke $r6.<java.io.PrintStream: void
        println(java.lang.String)>("min age of adult is 18")
$i0 = virtualinvoke $r3.<com.Person: int getAge()>()
$i1 = virtualinvoke $r2.<com.Person: int getAge()>()
if $i0 != $i1 goto $r6 = <java.lang.System: java.io.PrintStream out>
$r6 = <java.lang.System: java.io.PrintStream out>
virtualinvoke $r6.<java.io.PrintStream: void
        println(java.lang.String)>("They do not have the same age")
virtualinvoke $r3.<com.Person: void setGraduation(com.University)>($r1)
virtualinvoke $r3.<com.Person: void setGraduation(com.University)>($r5)
$r1 = virtualinvoke $r3.<com.Person: com.University getGraduation()>()
$r4 = virtualinvoke $r2.<com.Person: com.University getGraduation()>()
if $r1 != $r4 goto $r6 = <java.lang.System: java.io.PrintStream out>
$r6 = <java.lang.System: java.io.PrintStream out>
virtualinvoke $r6.<java.io.PrintStream: void
        println(java.lang.String)>("Their graduate is the same")
```

Thank you!