AjMutator: a Tool for the Mutation Analysis of Pointcut Descriptors

Romain Delamare[1] Benoit Baudry[1] Yves Le Traon[2]

[1]IRISA / INRIA
Rennes, France

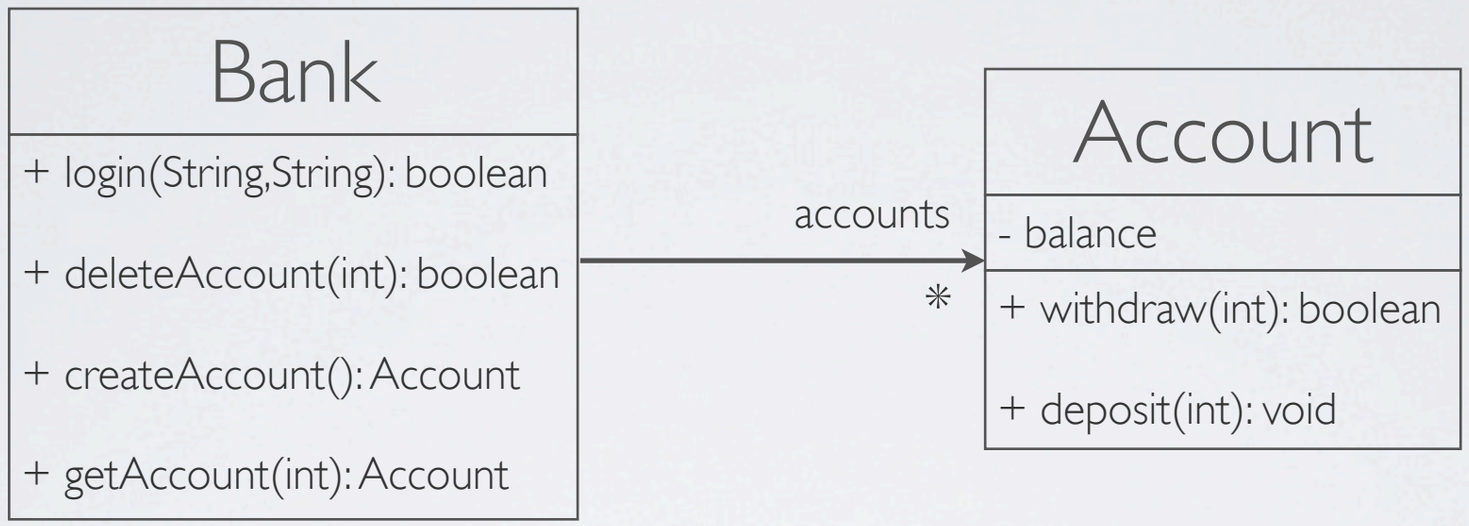
[2]Télécom Bretagne
Rennes, France

Aspect Oriented Programming

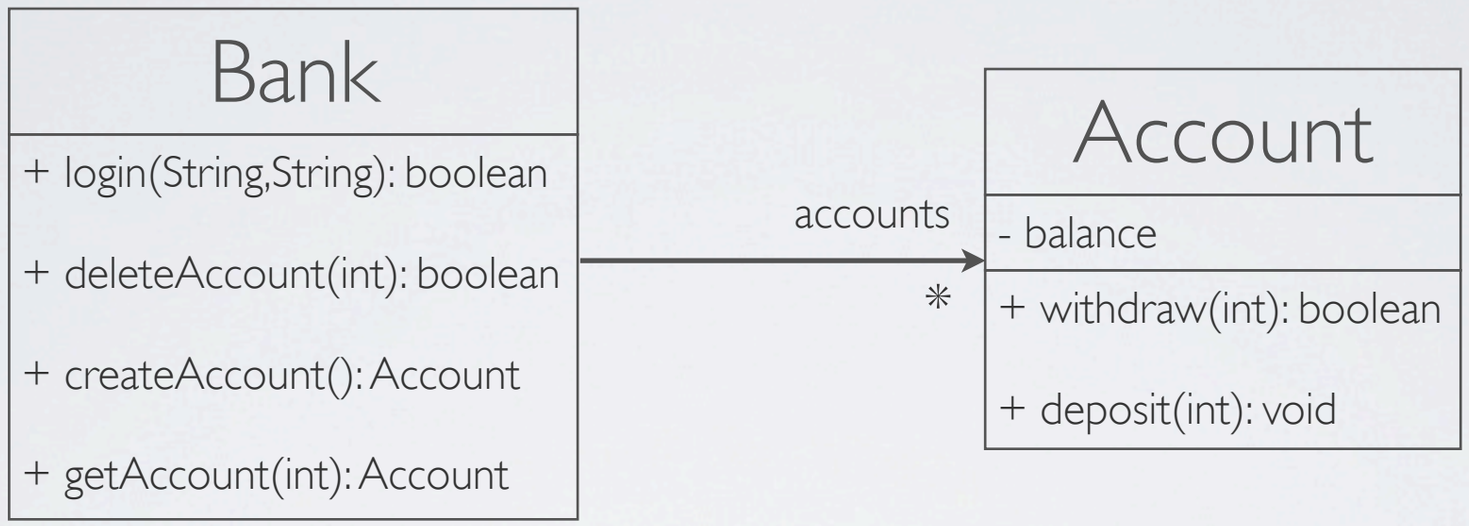
- The crosscutting concerns are separated from the core concern

- Aspect
  - Advice
  - Pointcut Descriptor (PCD)
    - Describes a set of joinpoints

Aspect Oriented Programming: example

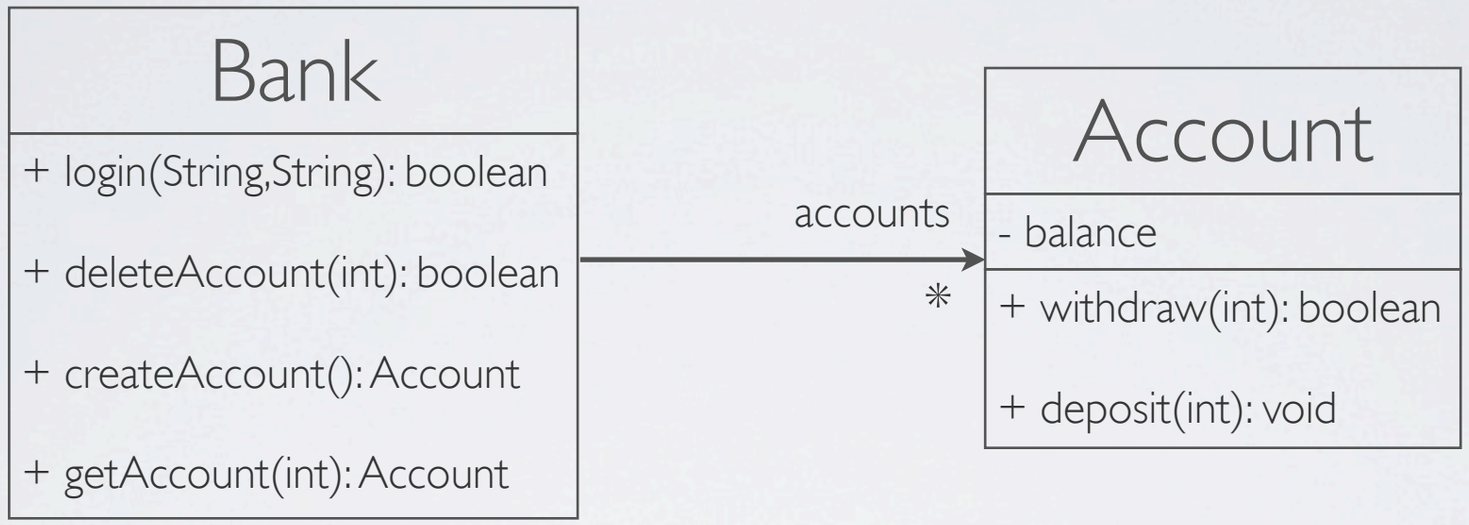


Aspect Oriented Programming: example



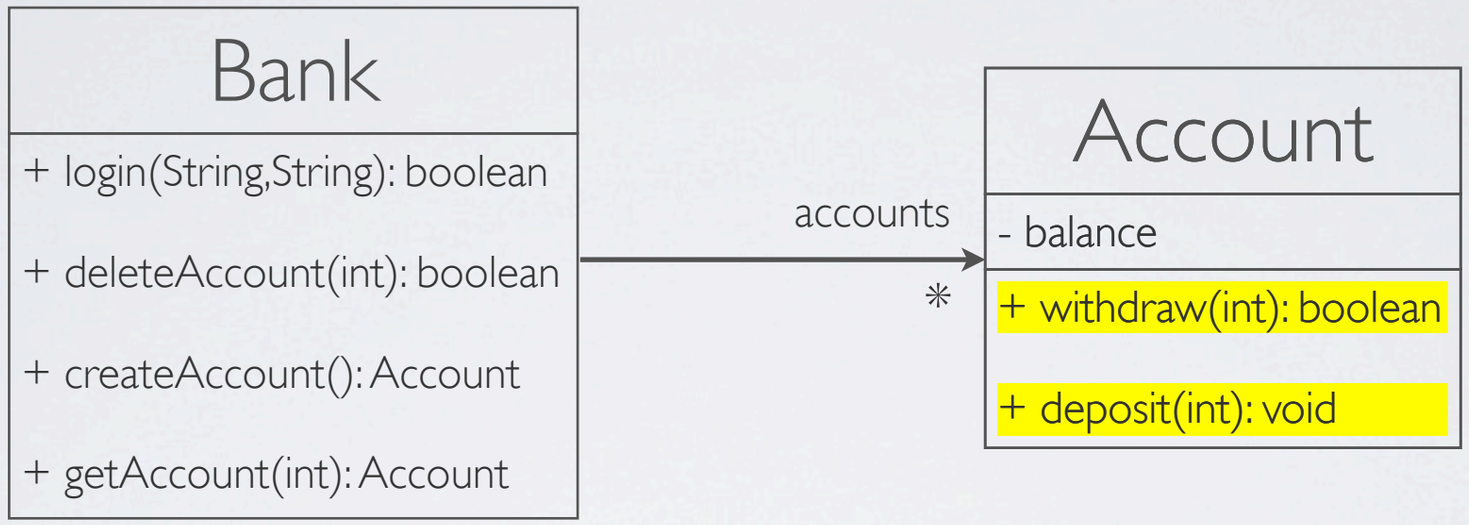

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```

Aspect Oriented Programming: example



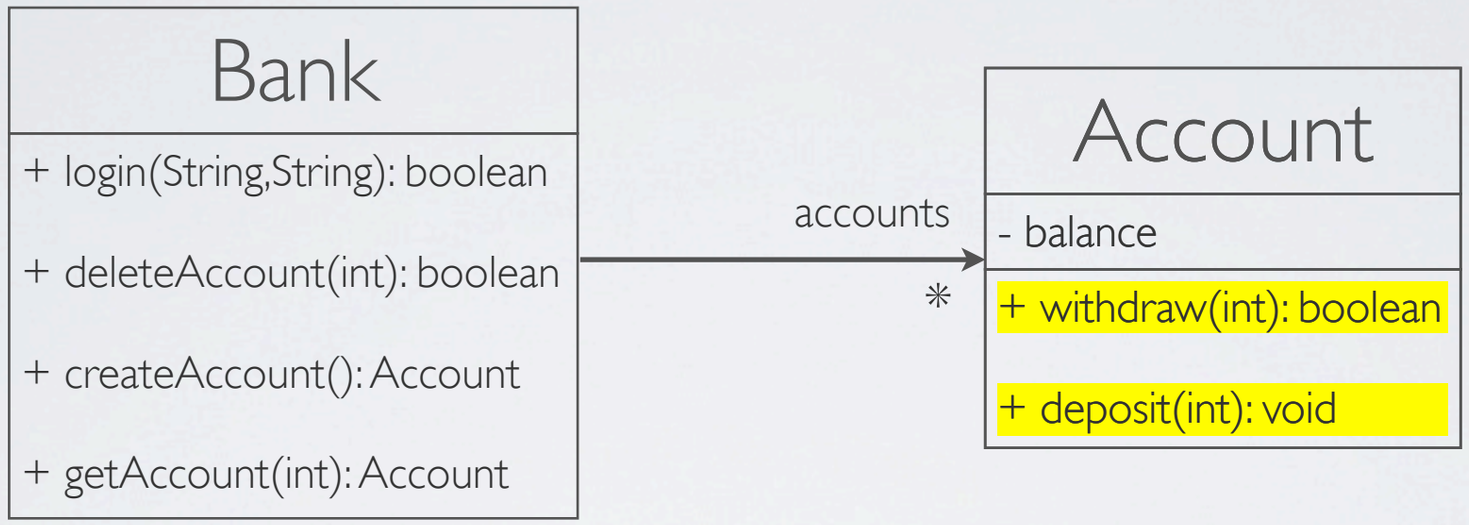

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```

Aspect Oriented Programming: example



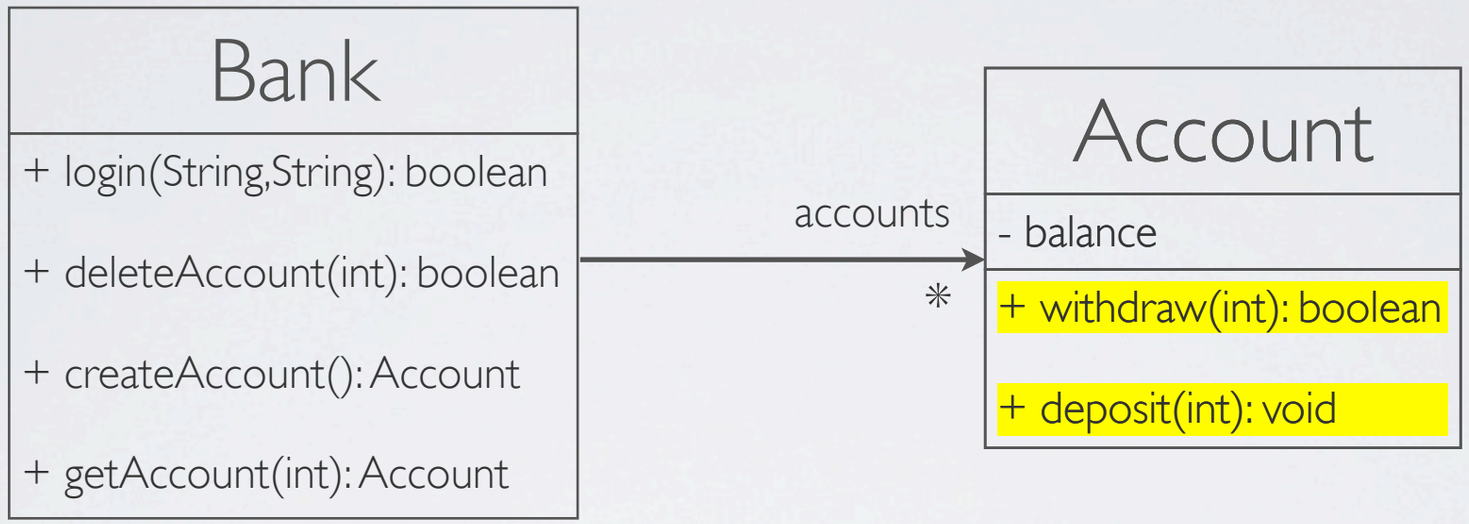

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```

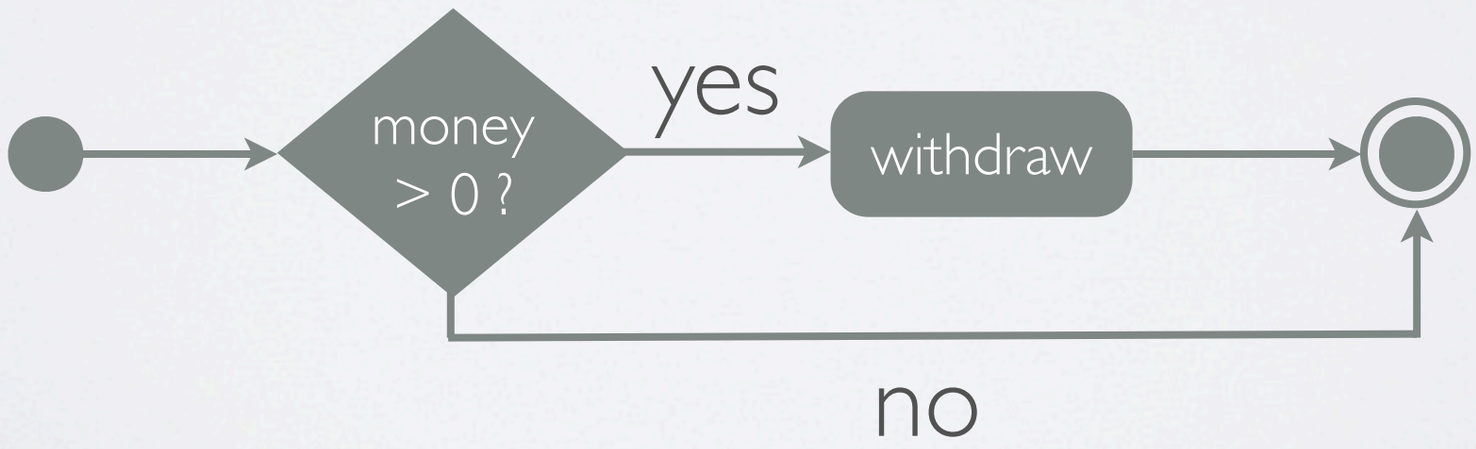
Aspect Oriented Programming: example



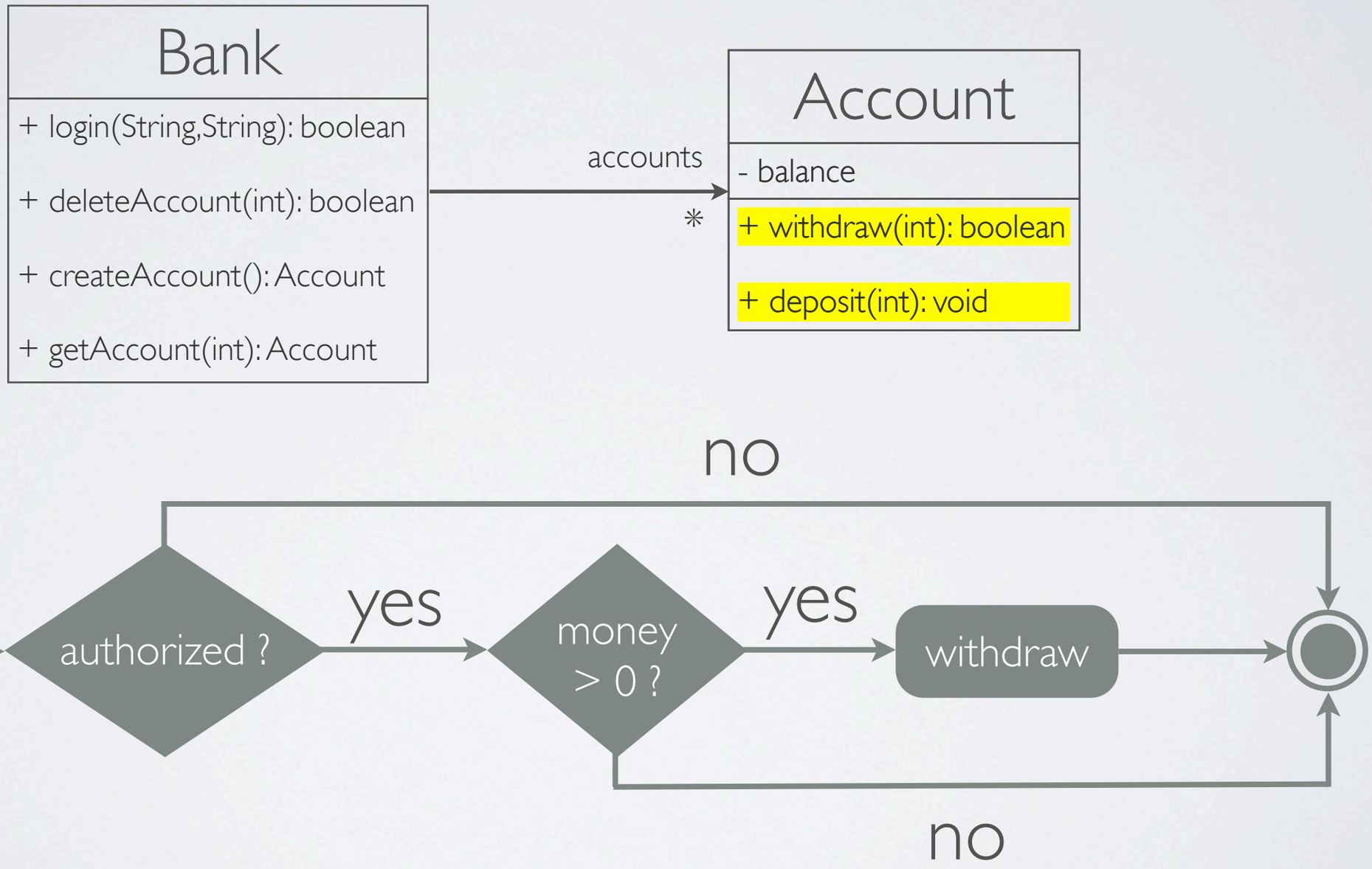

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```

Aspect Oriented Programming: example





Aspect Oriented Programming: example



Classes of Faults in the Pointcut Descriptor

Intended

Matched

Classes of Faults in the Pointcut Descriptor

correct PCD

Intended

Matched

Classes of Faults in the Pointcut Descriptor

correct PCD

neglected joinpoints

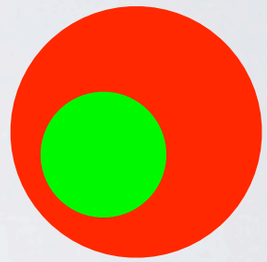
Intended

Matched

Classes of Faults in the Pointcut Descriptor

correct PCD

neglected joinpoints



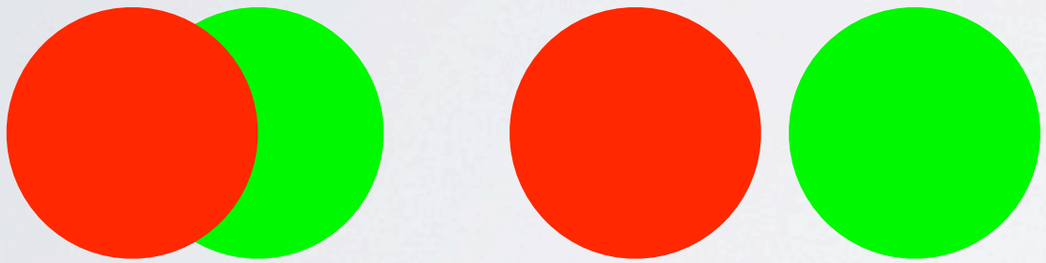
unintended joinpoints

Intended

Matched

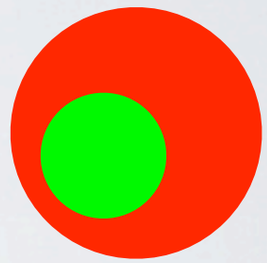
Classes of Faults in the Pointcut Descriptor

correct PCD



both
neglected and unintended

neglected
joinpoints



unintended
joinpoints

Intended

Matched

Classes of Faults in the Pointcut Descriptor

correct PCD

class 1

class 2

class 3

Intended

Matched

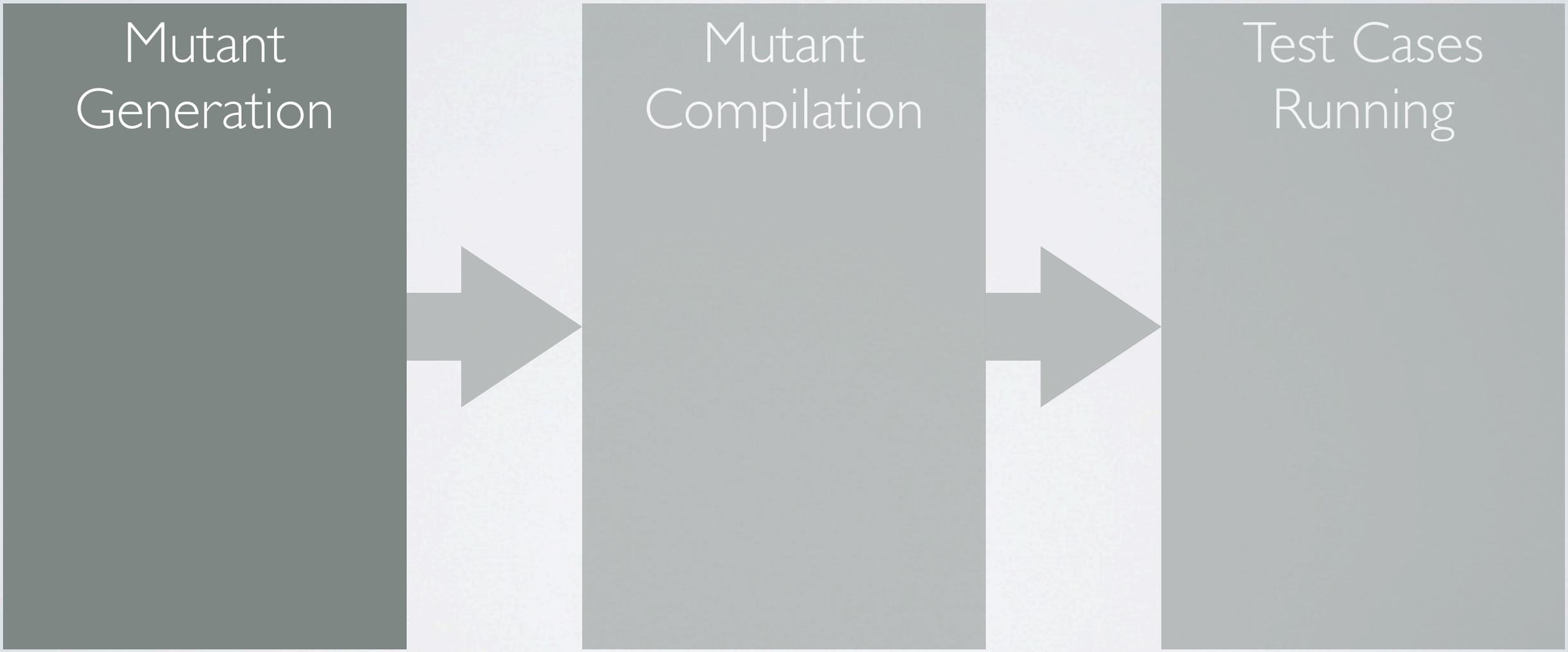
Mutant Pointcut Descriptor

- A PCD where a fault has been inserted
  - Selects a different set of joinpoints
- Equivalent mutant
  - Mutant that matches the same set of joinpoint

Mutant Pointcut Descriptor

- A PCD where a fault has been inserted
  - Selects a different set of joinpoints
- Equivalent mutant
  - Mutant that matches the same set of joinpoint
  - Equivalent mutants can be detected statically

AjMutator: Overview



AjMutator: Overview

Mutant Generation

- PCD parsing

Mutant Compilation

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD

Mutant Compilation

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant
- Classification

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant
- Classification
- Selection

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant
- Classification
- Selection

Test Cases Running

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant
- Classification
- Selection

Test Cases Running

- Test cases are executed on each mutant system

AjMutator: Overview

Mutant Generation

- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

- Compilation
  - A jar file for each mutant
- Classification
- Selection

Test Cases Running

- Test cases are executed on each mutant system
- Mutation score for the test suite

Mutant Generation

- An Abstract Syntax Tree (AST) for each PCD
- Mutation operators from [Ferrari *et al*., ICST'08]
- Implemented as AST visitors
  - Inserts the fault by modifying the AST
  - The AST is pretty-printed in a mutant source file
- New operators can added easily

Mutation Operators

Operator	Description
PCCC	Replaces a `cflow` by a `cflowbelow`, or the contrary
PCCE	Replaces a `call` by an `execution`, or the contrary
PCGS	Replaces a `get` by a `set`, or the contrary
PCLO	Changes the logical operators in a composition of PCDs
PCTT	Replaces a `this` by a `target`, or the contrary
POEC	Adds, removes or changes throwing clauses
POPL	Changes the parameter list
PSWR	Removes wildcards
PWAR	Removes annotation from type, field or method patterns
PWIW	Adds wildcards

Mutant Generation: problem

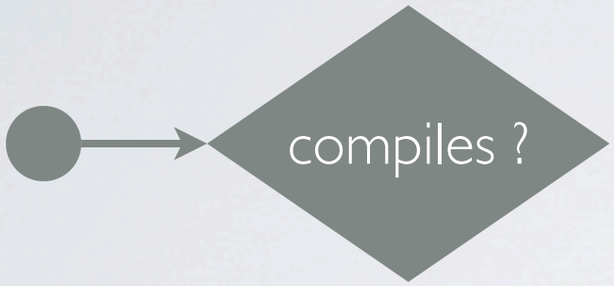
- What we want: mutants PCDs selecting different joinpoints

- What we do: modify the declaration of the PCD
- Problem:
  - Several different declarations can describe the same PCD
  - Thus we can have an equivalent mutant.

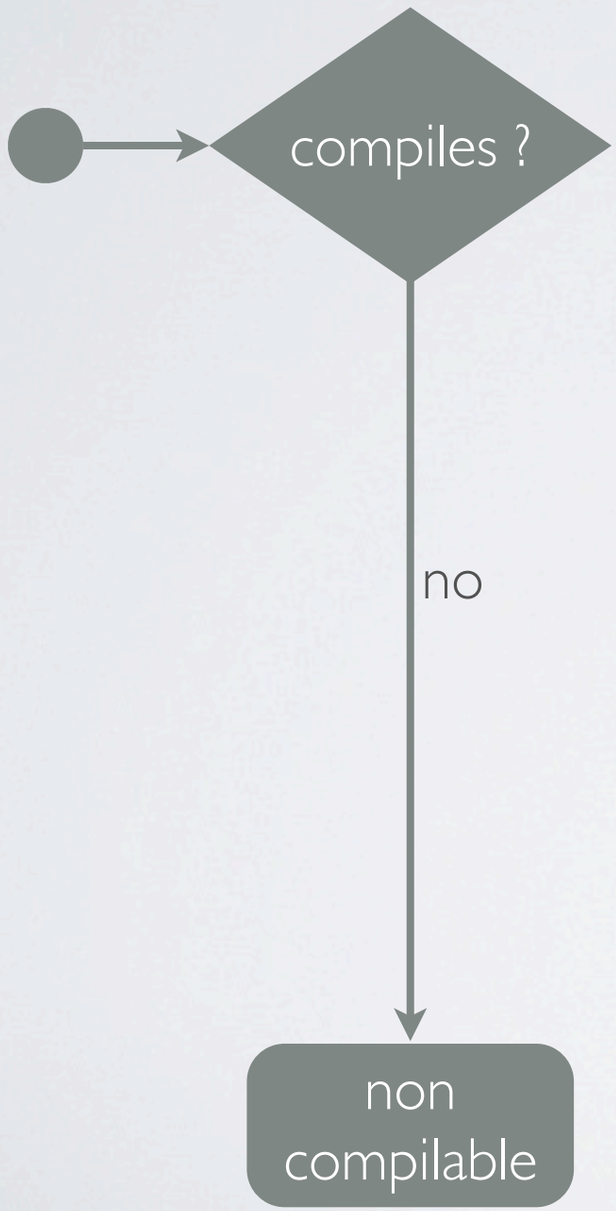
Mutant Compilation

- Each mutant is compiled
- If the compilation does not fail, the mutant is classified
  - Automatic classification, using the previous classification
- A selection of the mutant is made, depending on their class
  - The mutation analysis only considers selected mutants

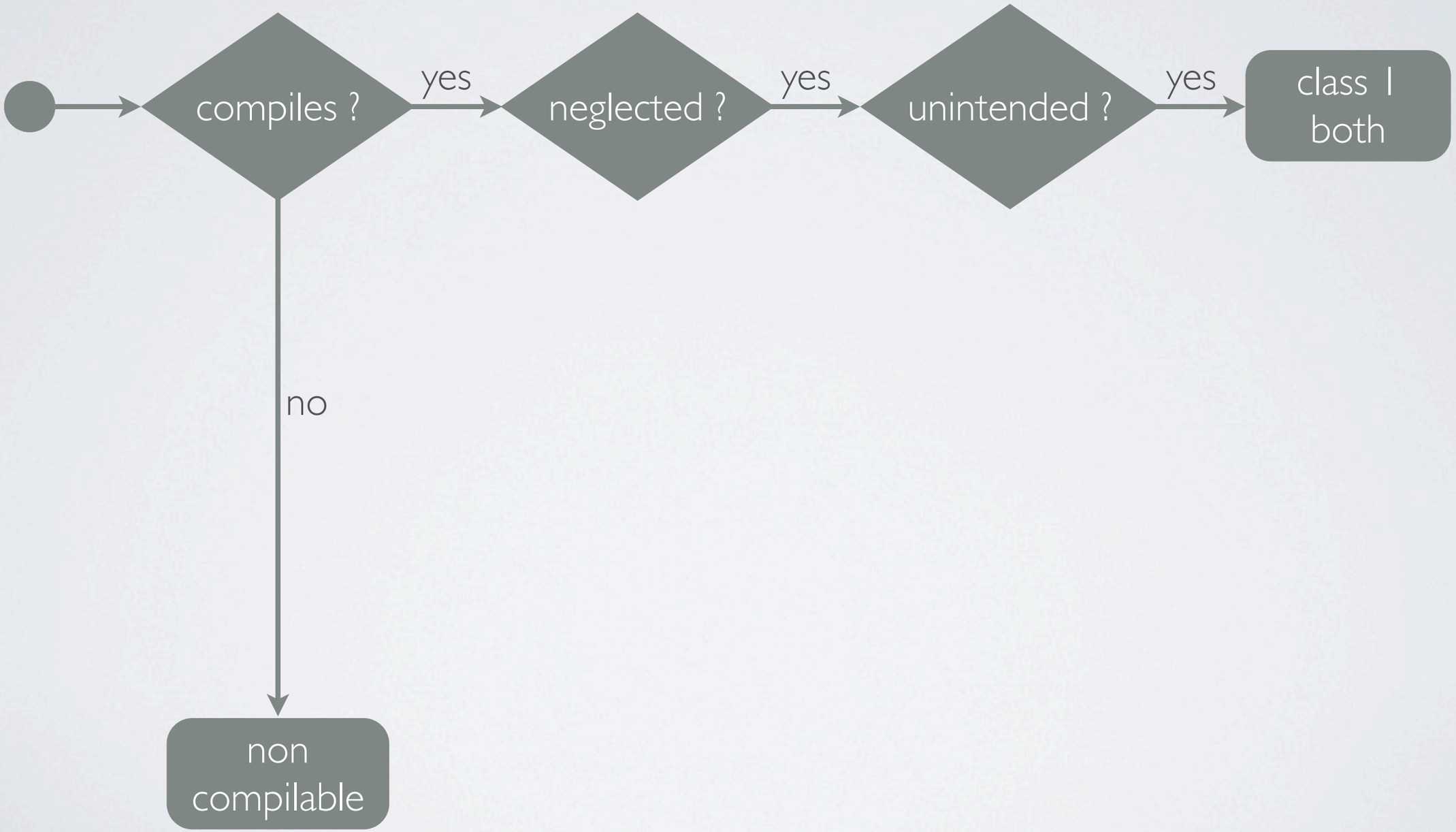
Automatic Classification and Selection of the Mutants



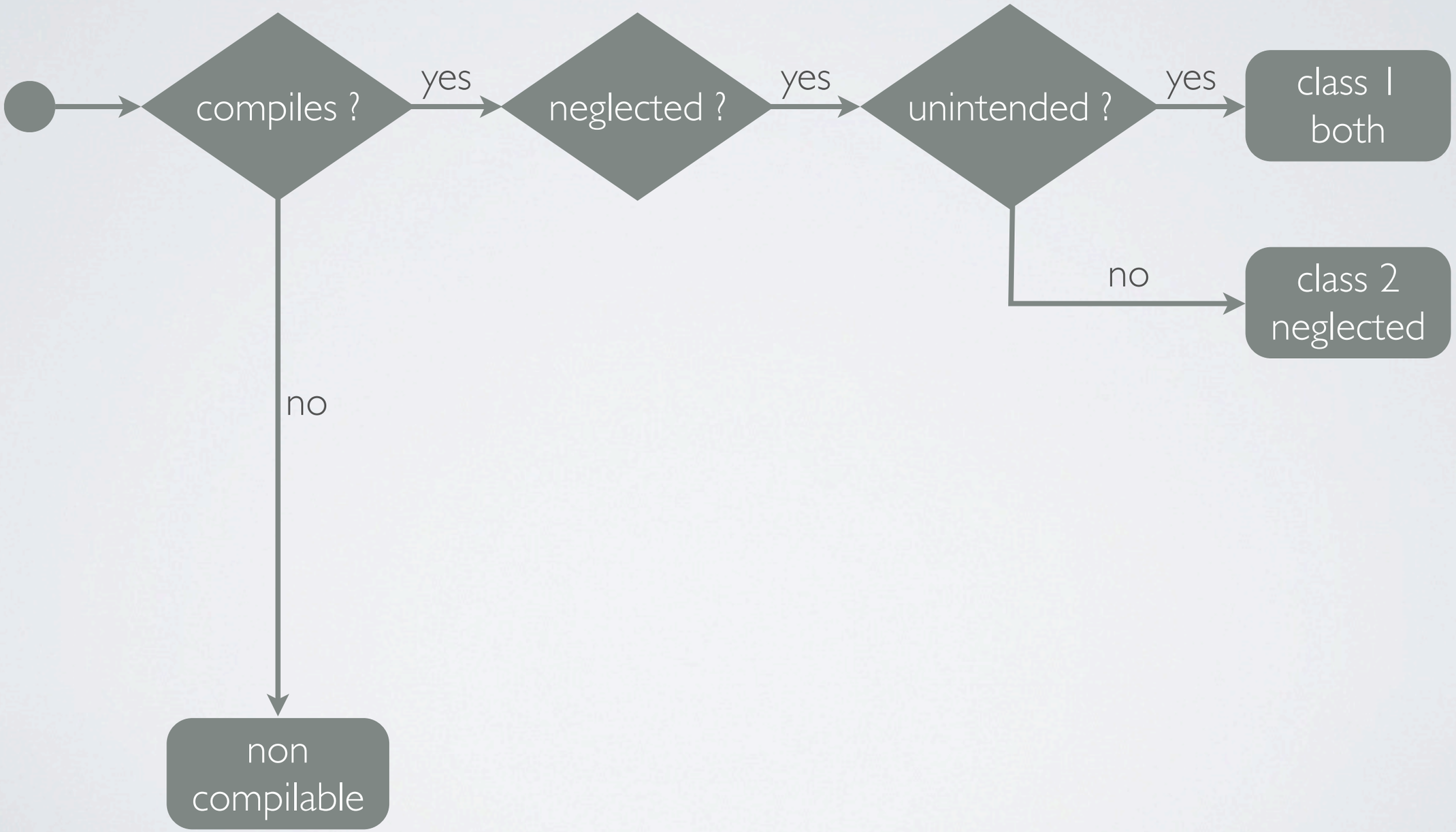
Automatic Classification and Selection of the Mutants



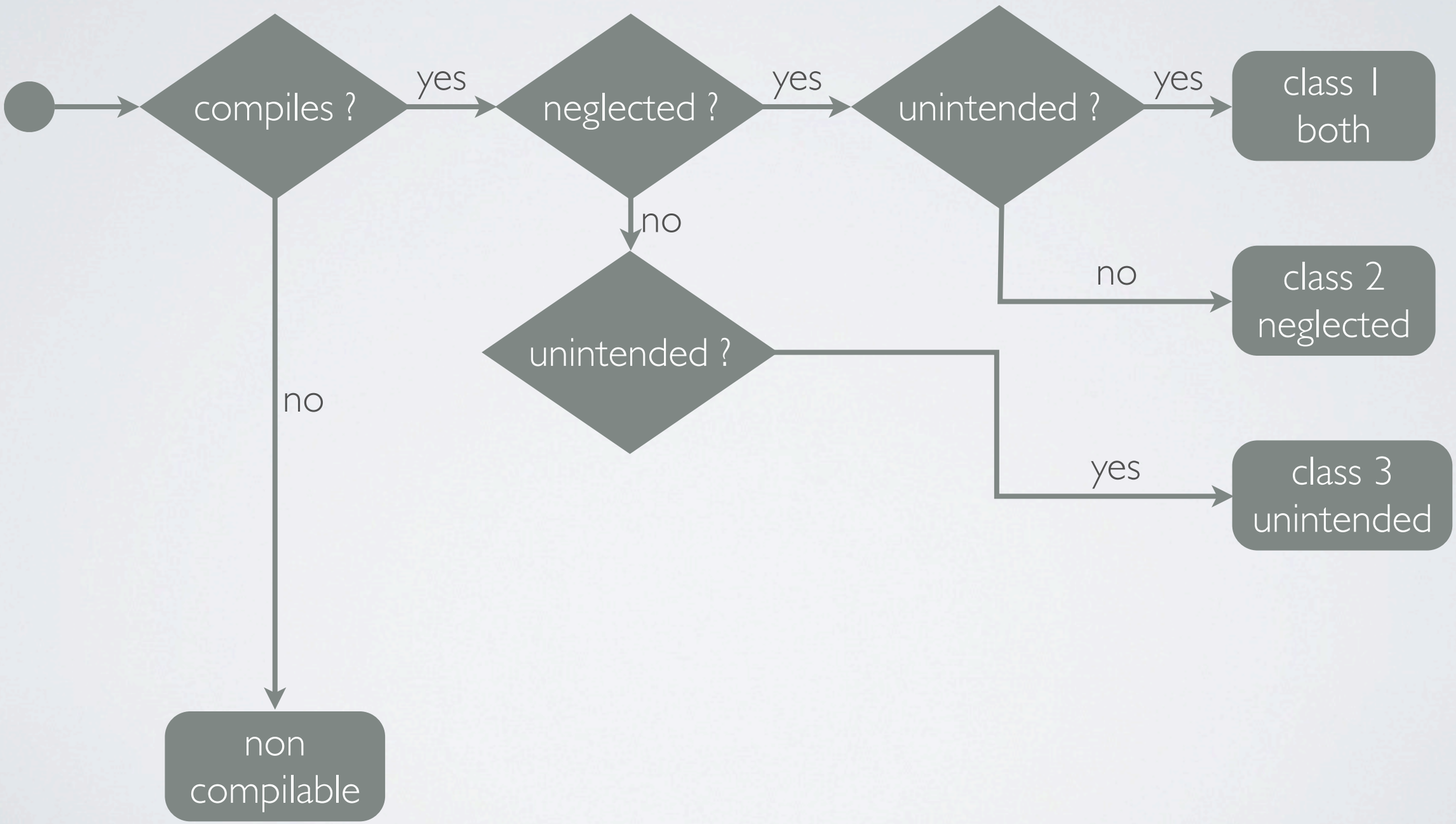
Automatic Classification and Selection of the Mutants



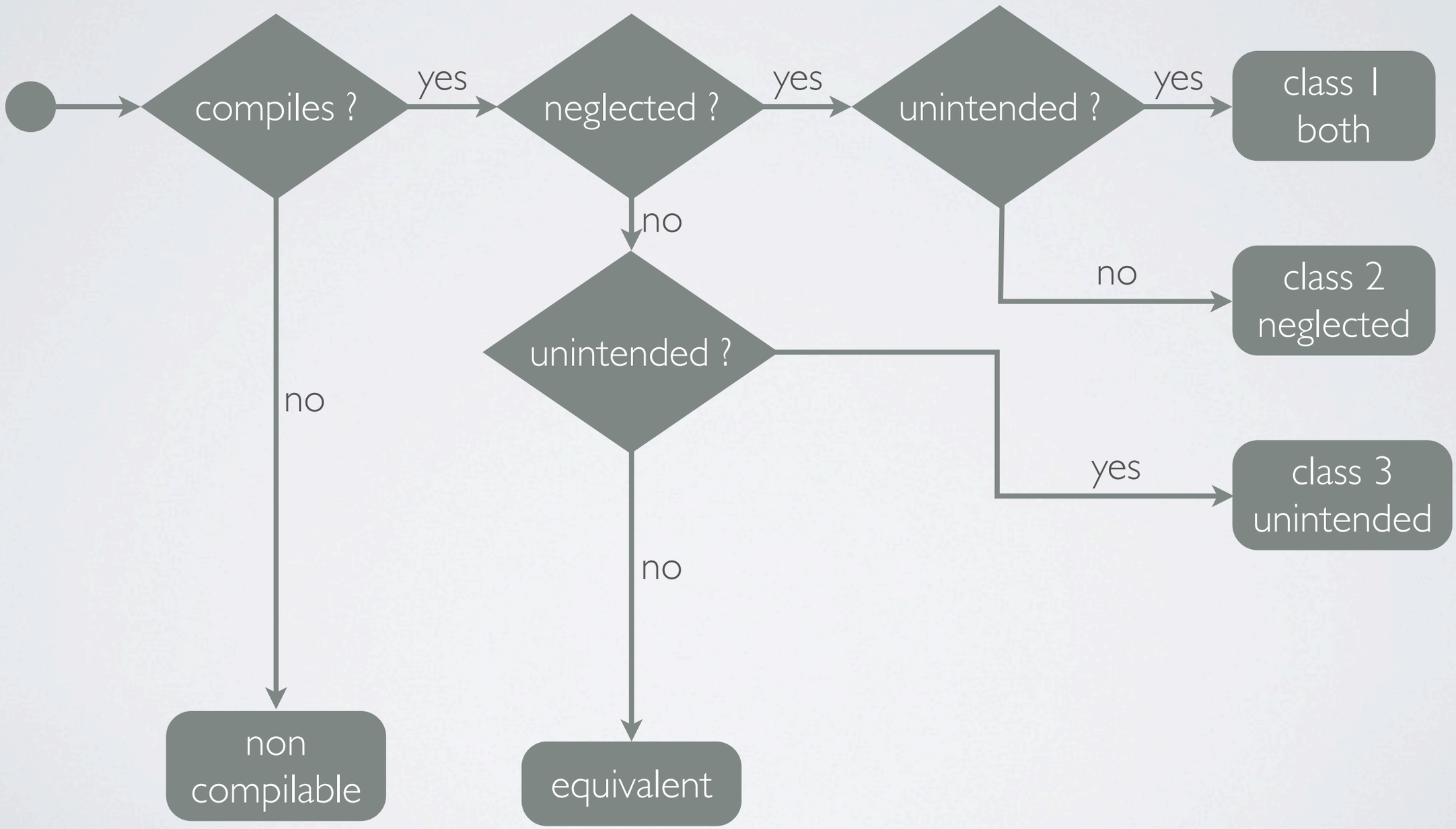
Automatic Classification and Selection of the Mutants



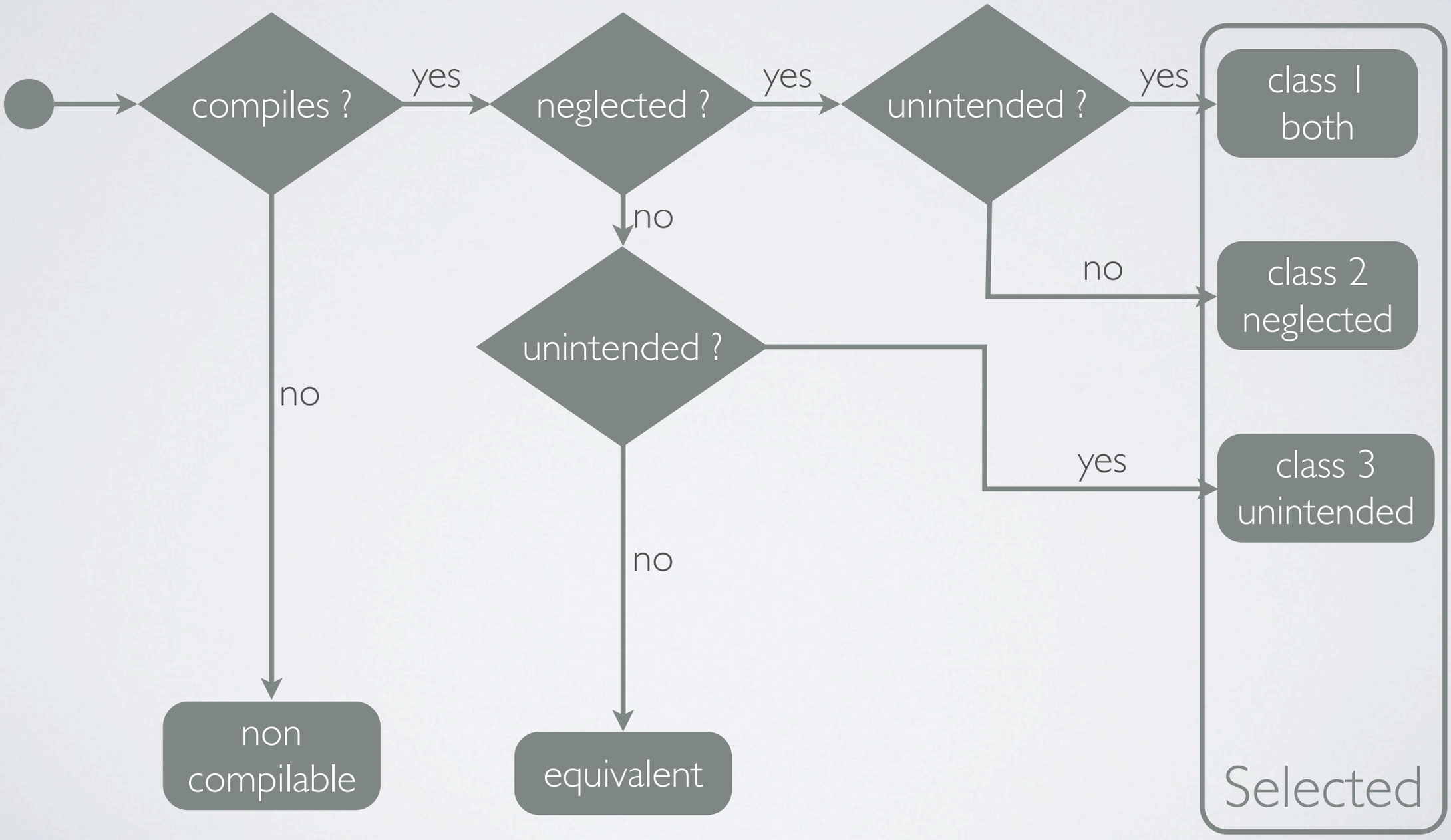
Automatic Classification and Selection of the Mutants



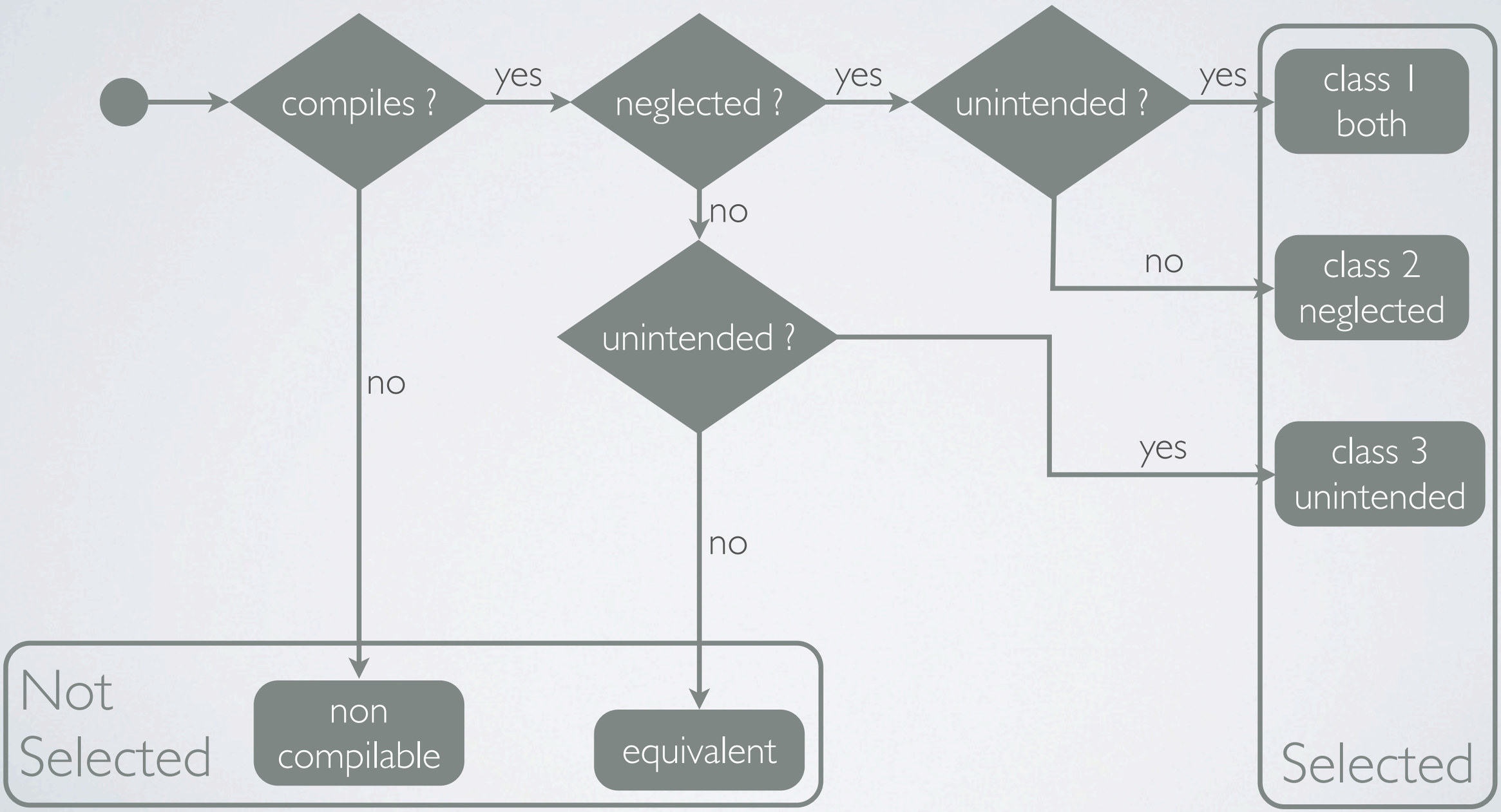
Automatic Classification and Selection of the Mutants



Automatic Classification and Selection of the Mutants



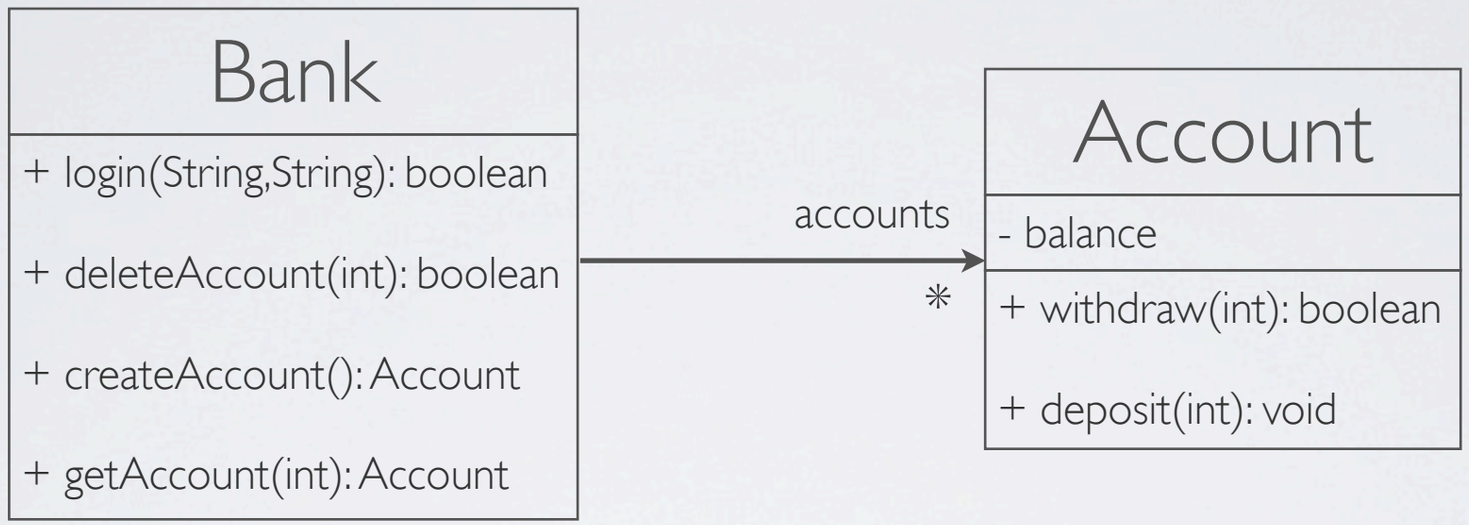
Automatic Classification and Selection of the Mutants



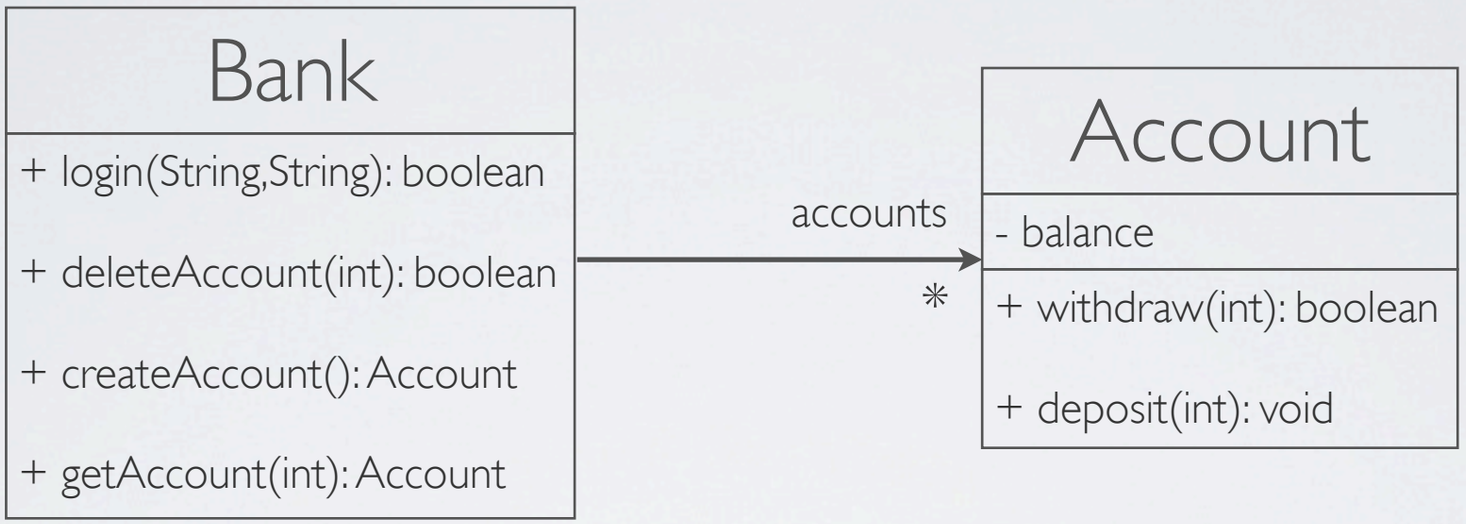
Static and Dynamic PCDs

- All PCDs have a static part
  - Worst case: all the joinpoints of the program
- Some PCDs have a dynamic part (Dynamic PCDs)
  - At runtime the dynamic part decides whether the advice is executed or not (restriction of the joinpoints)
  - At compile time the set of joinpoints matched by a dynamic PCD can only be over-approximated

Dynamic PCD: example

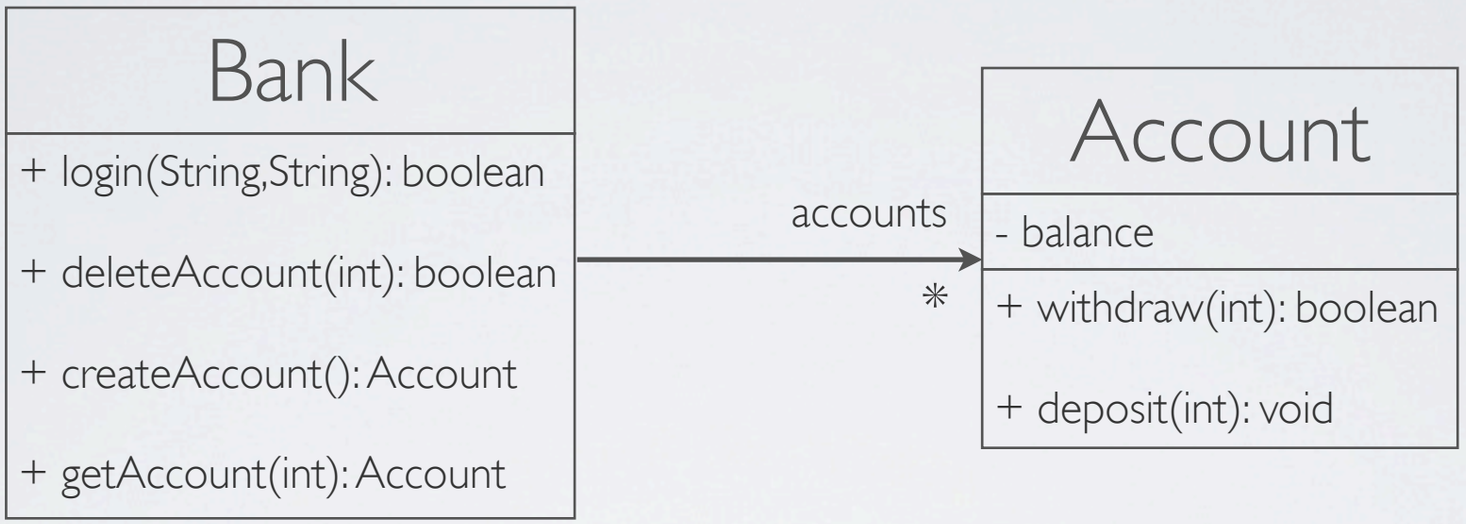


Dynamic PCD: example




```
pointcut controlledAccess(): get(Account.balance) &&
                             cflow(execution(* Auction.withdraw(int)))
```

Dynamic PCD: example

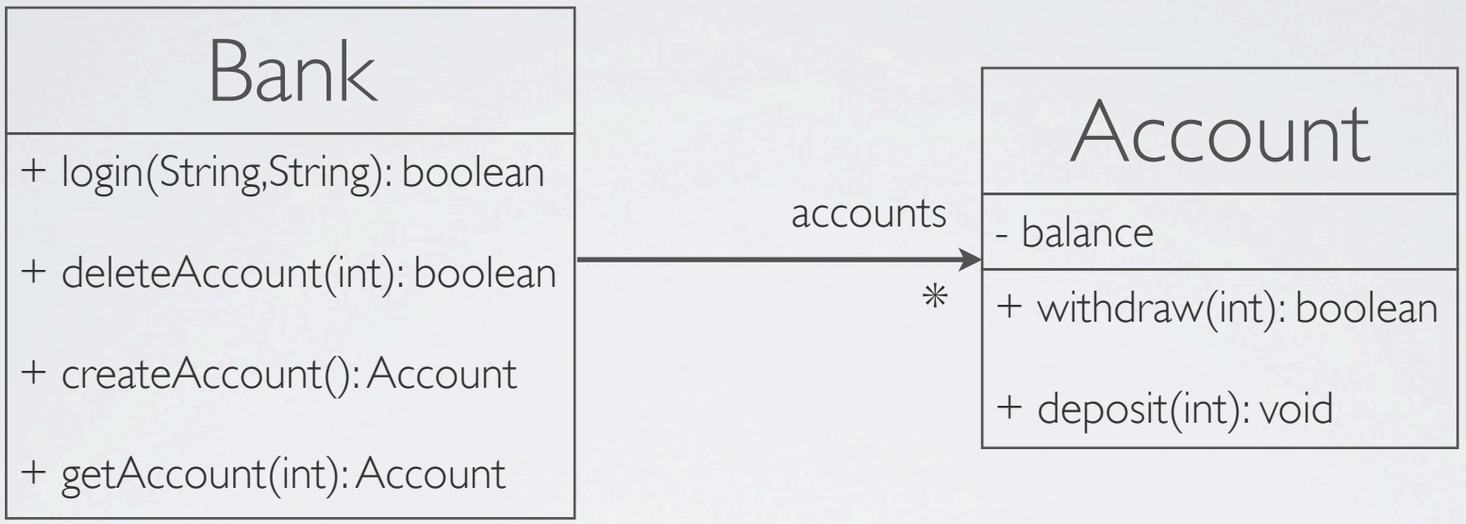



```
pointcut controlledAccess(): get(Account.balance) &&
                             cflow(execution(* Auction.withdraw(int)))
```

matched

```
public boolean withdraw(int amount) {
    if(balance>0) {
        // ...
    }
```

Dynamic PCD: example




```
pointcut controlledAccess(): get(Account.balance) &&
                             cflow(execution(* Auction.withdraw(int)))
```

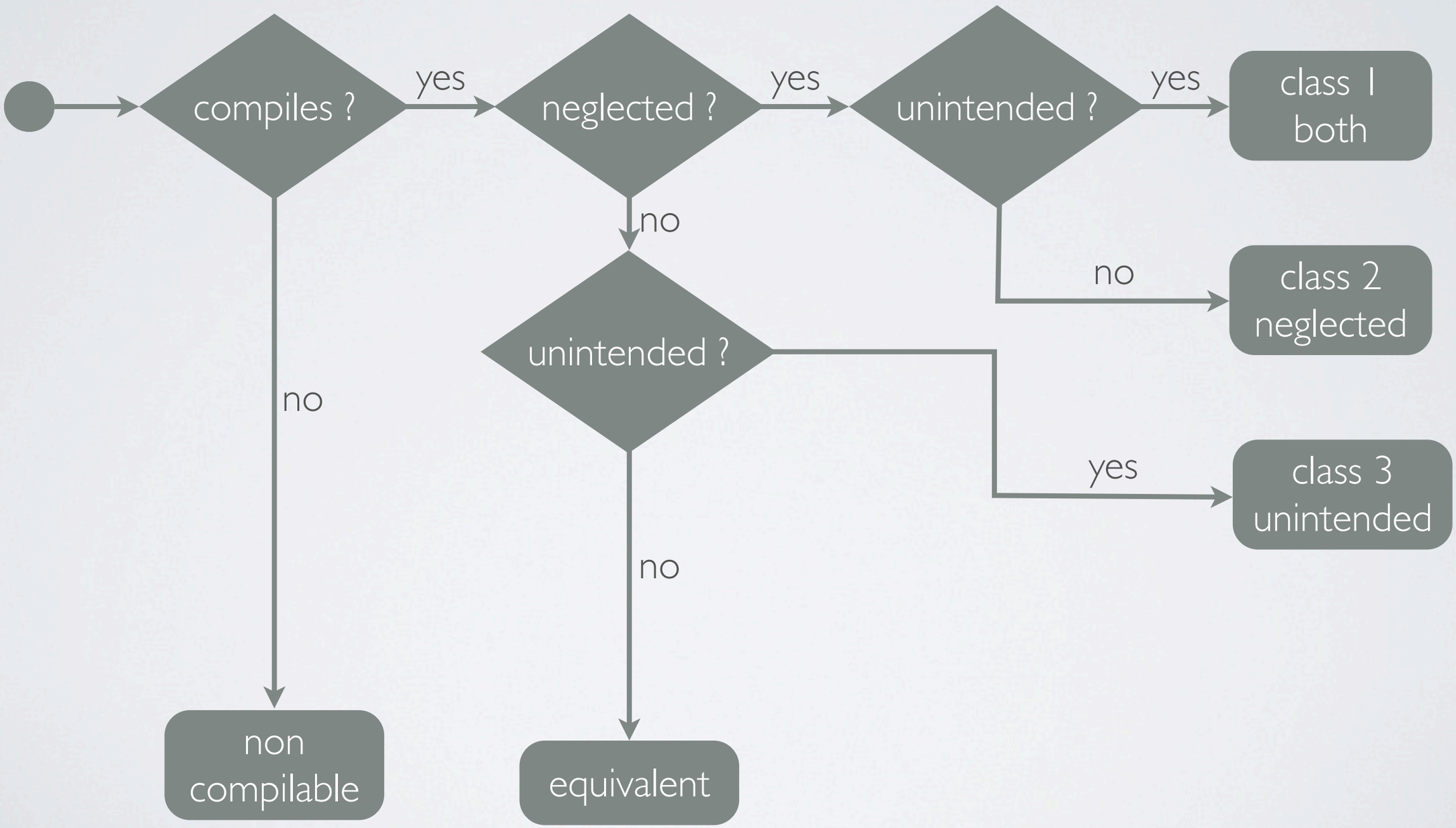
matched →

```
public boolean withdraw(int amount) {
    if(balance>0) {
        // ...
    }
```

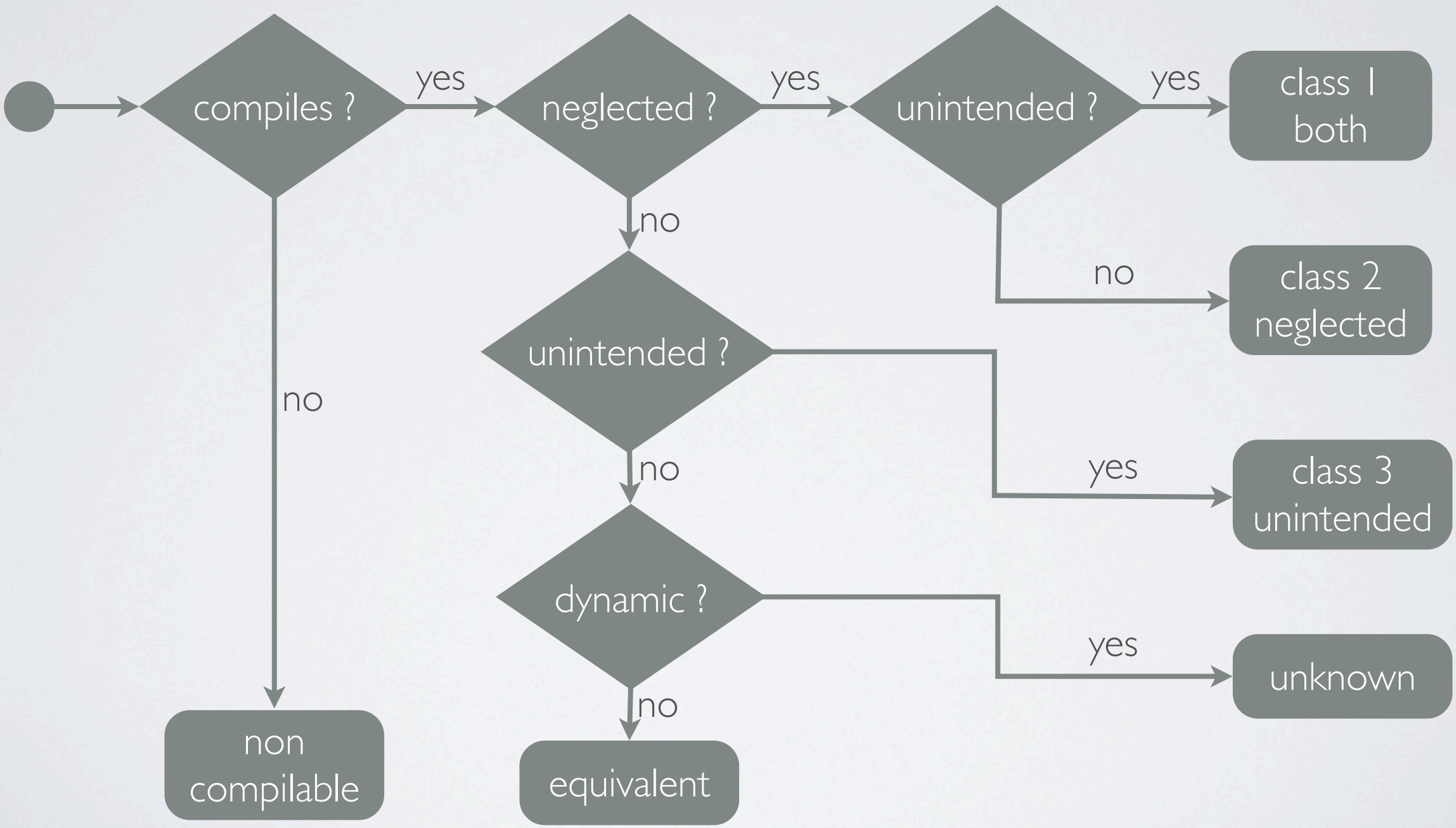
not matched →

```
public void deposit(int amount) {
    if(balance>0) {
        // ...
    }
```

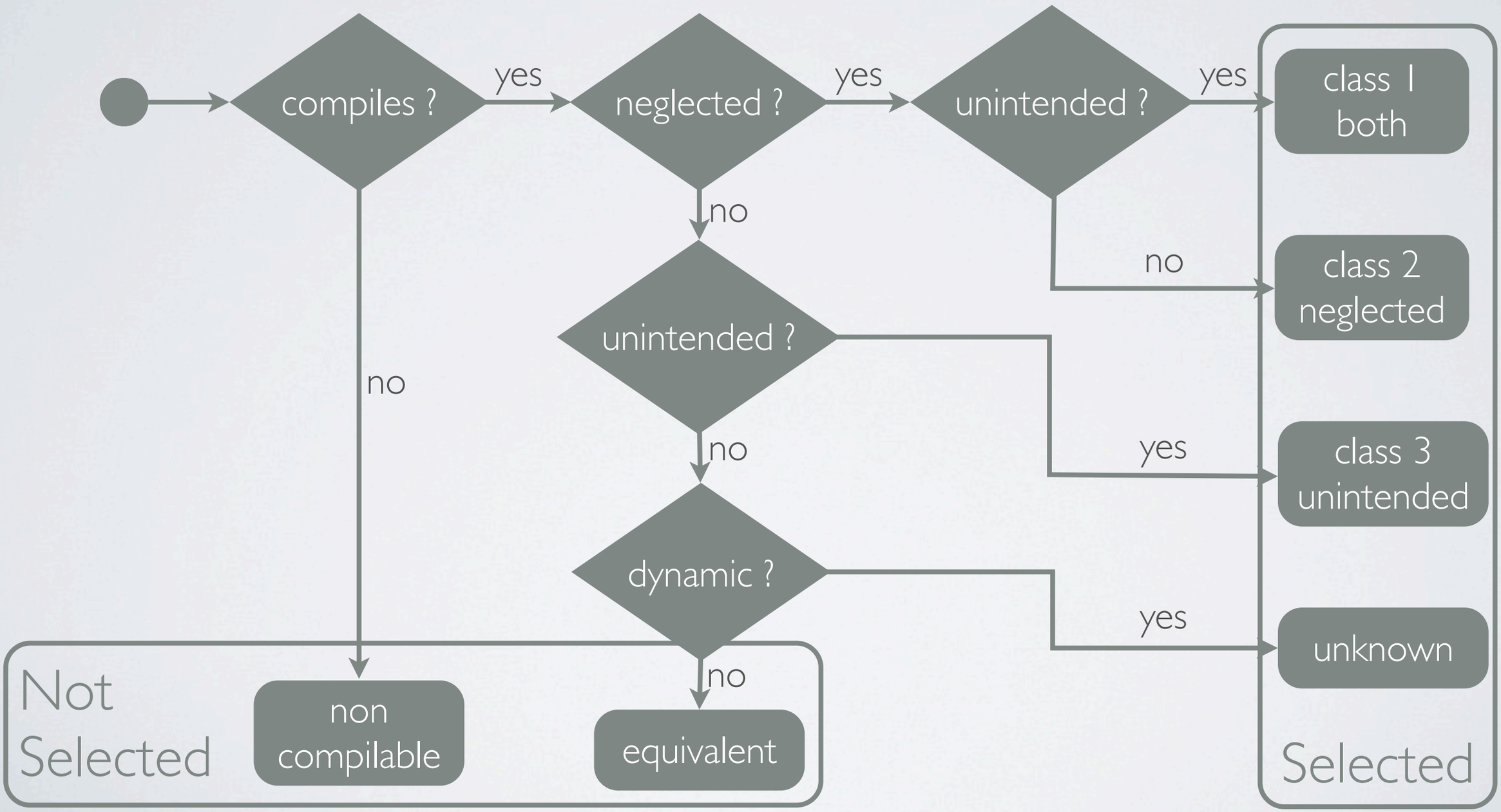
Classification and Selection with Dynamic PCDs



Classification and Selection with Dynamic PCDs



Classification and Selection with Dynamic PCDs



Execution of the Test Suite

- The test suite is executed on each mutant system
  - JUnit test suite
    - All tests pass on the original system
- A mutant is killed if at least one test case fails
  - Qualification of the JUnit oracle
- A mutation score for the test suite

Conclusion

- AjMutator, a tool for the mutation analysis of PCDs
  - Operators insert faults in the PCDs
    - Mutant are compiled, classified, and selected automatically
      - Automatic detection of the equivalent mutant in most cases
    - Execution of a Test Suite
- http://www.irisa.fr/triskell/softwares-fr/protos/AjMutator/

Evaluation on HealthWatcher

Class	Number of Mutants
1 (both)	55
2 (neglected)	50
3 (unintended)	129
unknown	65
Total Selected	299
Equivalent	296
Non-Compilable	90
Total	685