Instant Code Clone Search
(비슷한 소스 코드 **빨리** 찾기)

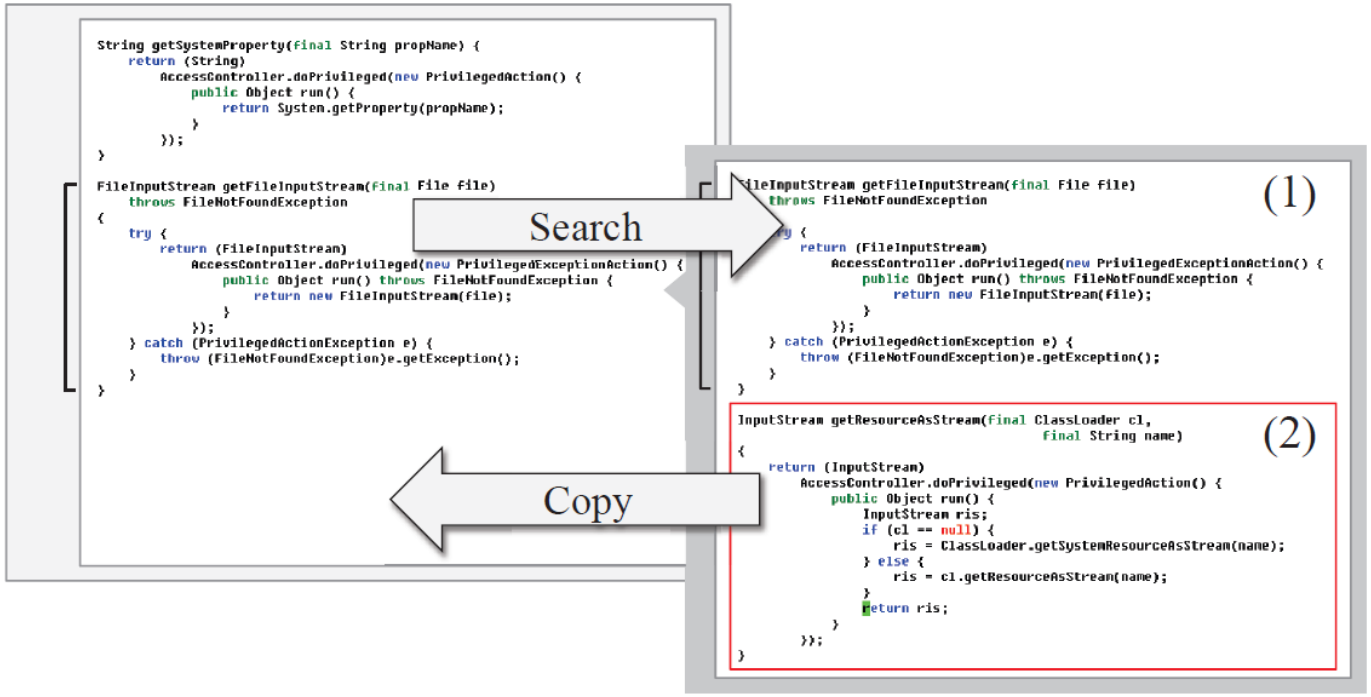
이상훈
Information and Database Systems 연구실
POSTECH

POSTECH

Motivation

- Code clone search
  - CCFinder (2002) : 토큰 기반의 코드 클론 검색
  - Deckard (2007) : AST에 근사한 벡터 기반 코드 클론 검색

- 개발 세션 중간에 코드 검색



Background

- Deckard (2007)
  - AST를 벡터들의 집합으로 근사하고 LSH를 사용하여 구조적 코드 클론을 검색

- R-트리
  - 다차원 정보를 인덱싱하기 위한 B 트리와 유사한 데이터 구조
  - 가까운 거리의 데이터를 MBR (Minimum Bounding Rectangle)로 묶어 저장
  - k-NN (Nearest Neighbor) 질의 가능

- Challenge
  - 자바 SE JDK 1.6u13
    - 7,195 파일, 2,075,573 라인, 36,658 벡터
  - 자바 오픈 소스 프로젝트 492개
    - 288,846 파일, 54,709,384 라인, 612,926 벡터
  - 고차원 벡터 데이터베이스
    - 261 차원

Dimensionality Reduction

- Goal
  - 수백 차원의 데이터를 다룰 수 있는 수의 차원으로 축소
  - Lower-bounding property

$$\|v_i', v_j'\| \leq \|v_i, v_k\|$$

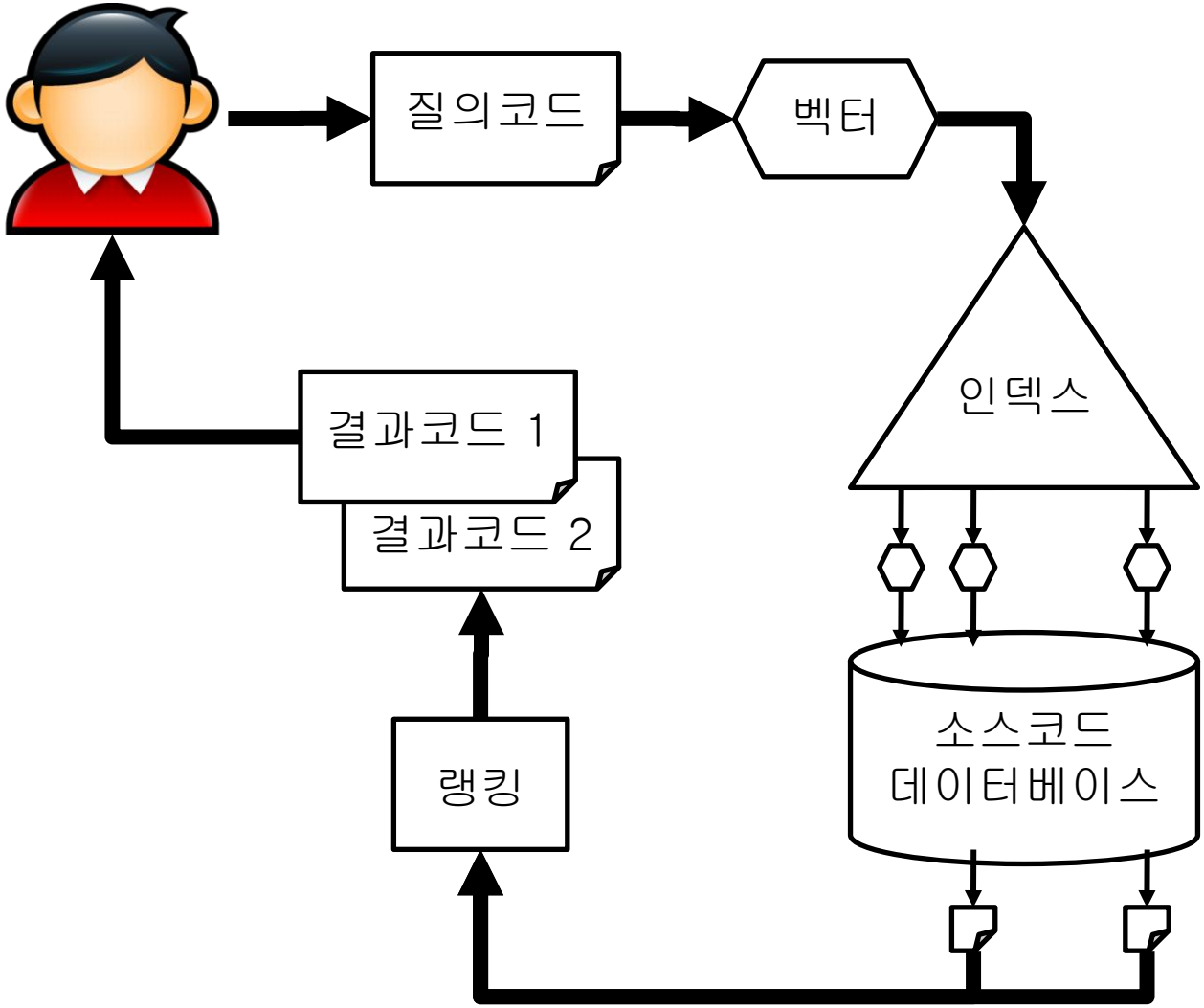
  - 두 벡터의 거리 관계를 가능한 보존

$$\Delta = \sum_{\forall i, \forall j, i \neq j} \delta_{i,j} = \sum_{\forall i, \forall j, i \neq j} \|v_i, v_j\| - \|v_i', v_j'\|$$

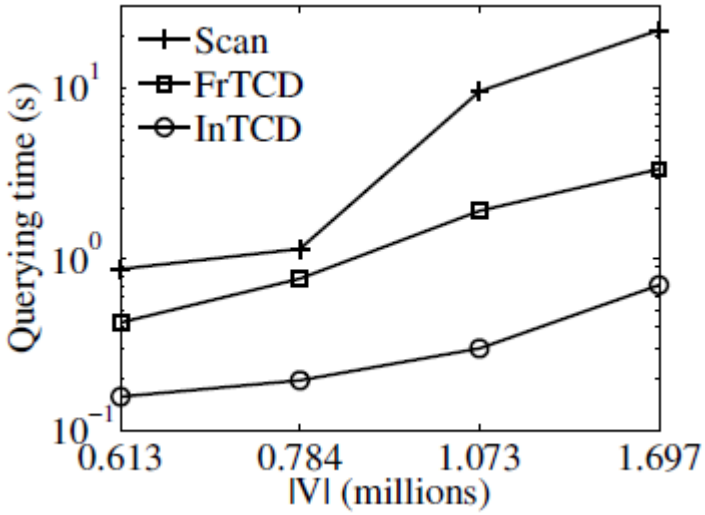
- NP-Hard

- 휴리스틱
  - 각 차원 내 값들 중 가장 편차가 큰 10개 차원을 선택

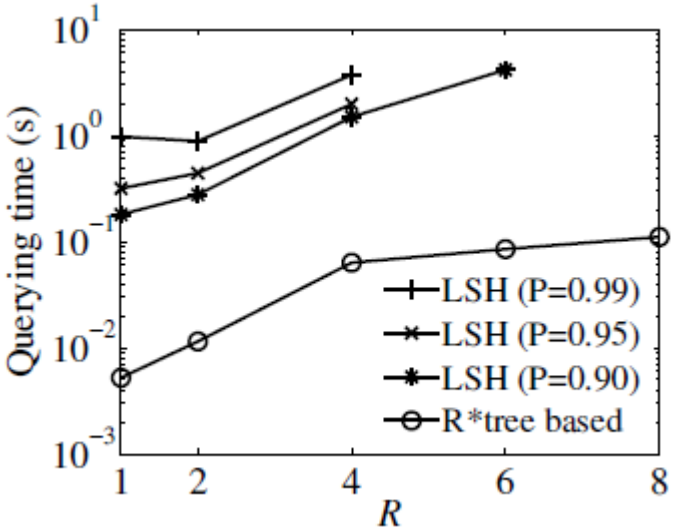
Process Overview



Performance



Querying time
for varying number of vectors, |V|



Querying time
for varying range, R

Q&A

- 이무웅, 노종원, 황승원, 김성훈. Instant Code Clone Search. FSE 2010

- 감사합니다.

Backup Slides

```java
String getSystemProperty(final String propName) {
    return (String)
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                return System.getProperty(propName);
            }
        });
}

FileInputStream getFileInputStream(final File file)
    throws FileNotFoundException
{
    try {
        return (FileInputStream)
            AccessController.doPrivileged(new PrivilegedExceptionAction() {
                public Object run() throws FileNotFoundException {
                    return new FileInputStream(file);
                }
            });
    } catch (PrivilegedActionException e) {
        throw (FileNotFoundException)e.getException();
    }
}
```

Search

Copy

(1)

```java
FileInputStream getFileInputStream(final File file)
    throws FileNotFoundException
{
    try {
        return (FileInputStream)
            AccessController.doPrivileged(new PrivilegedExceptionAction() {
                public Object run() throws FileNotFoundException {
                    return new FileInputStream(file);
                }
            });
    } catch (PrivilegedActionException e) {
        throw (FileNotFoundException)e.getException();
    }
}
```

(2)

```java
InputStream getResourceAsStream(final ClassLoader cl,
                                final String name)
{
    return (InputStream)
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                InputStream ris;
                if (cl == null) {
                    ris = ClassLoader.getSystemResourceAsStream(name);
                } else {
                    ris = cl.getResourceAsStream(name);
                }
                return ris;
            }
        });
}
```

```java
public void
setSourceBands(int[] sourceBands) {
  if (sourceBands == null) {
    this.sourceBands = null;
  } else {
    int numBands = sourceBands.length;
    for (int i = 0; i < numBands; i++) {
      int band = sourceBands[i];
      if (band < 0) {
        throw new IllegalArgumentException(
          "Band value < 0!");
      }
      for (int j = i + 1; j < numBands; j++) {
        if (band == sourceBands[j]) {
          throw new IllegalArgumentException(
            "Duplicate band value!");
        }
      }
    }
    this.sourceBands =
      (int[])(sourceBands.clone());
  }
}
```

SEARCH

```java
public void
setDestinationBands(int[] destinationBands) {
  if (destinationBands == null) {
    this.destinationBands = null;
  } else {
    int numBands = destinationBands.length;
    for (int i = 0; i < numBands; i++) {
      int band = destinationBands[i];
      if (band < 0) {
        throw new IllegalArgumentException(
          "Band value < 0!");
      }
      for (int j = i + 1; j < numBands; j++) {
        if (band == destinationBands[j]) {
          throw new IllegalArgumentException(
            "Duplicate band value!");
        }
      }
    }
    this.destinationBands =
      (int[])destinationBands.clone();
  }
}
```

COPY

```java
try {
  Enumeration mycomps = getPrefix(posn);
  while (mycomps.hasMoreElements()) {
    String my = (String)mycomps.nextElement();
    String his = (String)prefix.nextElement();
    if (syntaxTrimBlanks) {
      my = my.trim();
      his = his.trim();
    }
```

SEARCH

```java
try {
  Enumeration mycomps = getSuffix(startIndex);
  while (mycomps.hasMoreElements()) {
    String my = (String)mycomps.nextElement();
    String his = (String)suffix.nextElement();
    if (syntaxTrimBlanks) {
      my = my.trim();
      his = his.trim();
    }
```

```java
    if (syntaxCaseInsensitive) {
      if (!(my.equalsIgnoreCase(his)))
        return false;
    } else {
      if (!(my.equals(his)))
        return false;
    }
  }
} catch (NoSuchElementException e) {
  return false;
}
return true;
```

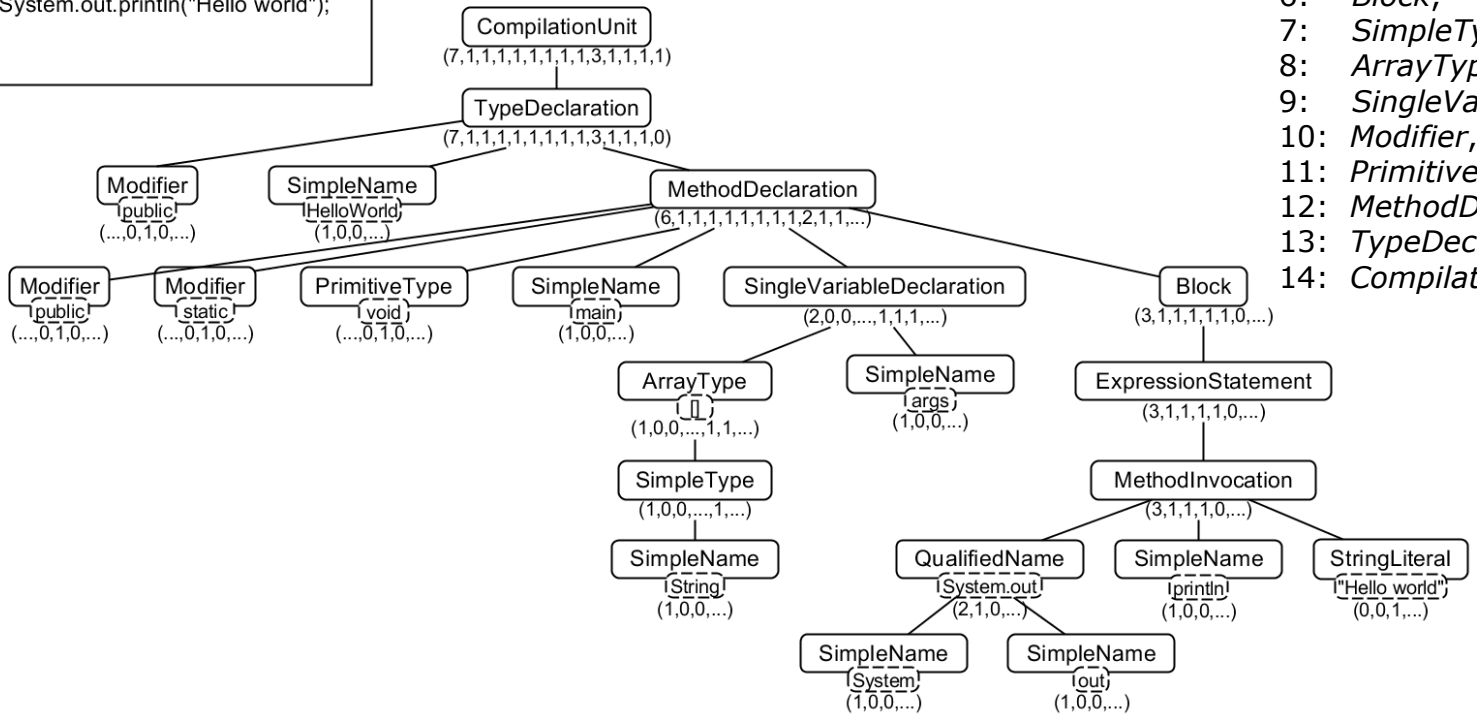
COPY

```java
    if (syntaxCaseInsensitive) {
      if (!(my.equalsIgnoreCase(his)))
        return false;
    } else {
      if (!(my.equals(his)))
        return false;
    }
  }
} catch (NoSuchElementException e) {
  return false;
}
return true;
```

Deckard

- Vector generation

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

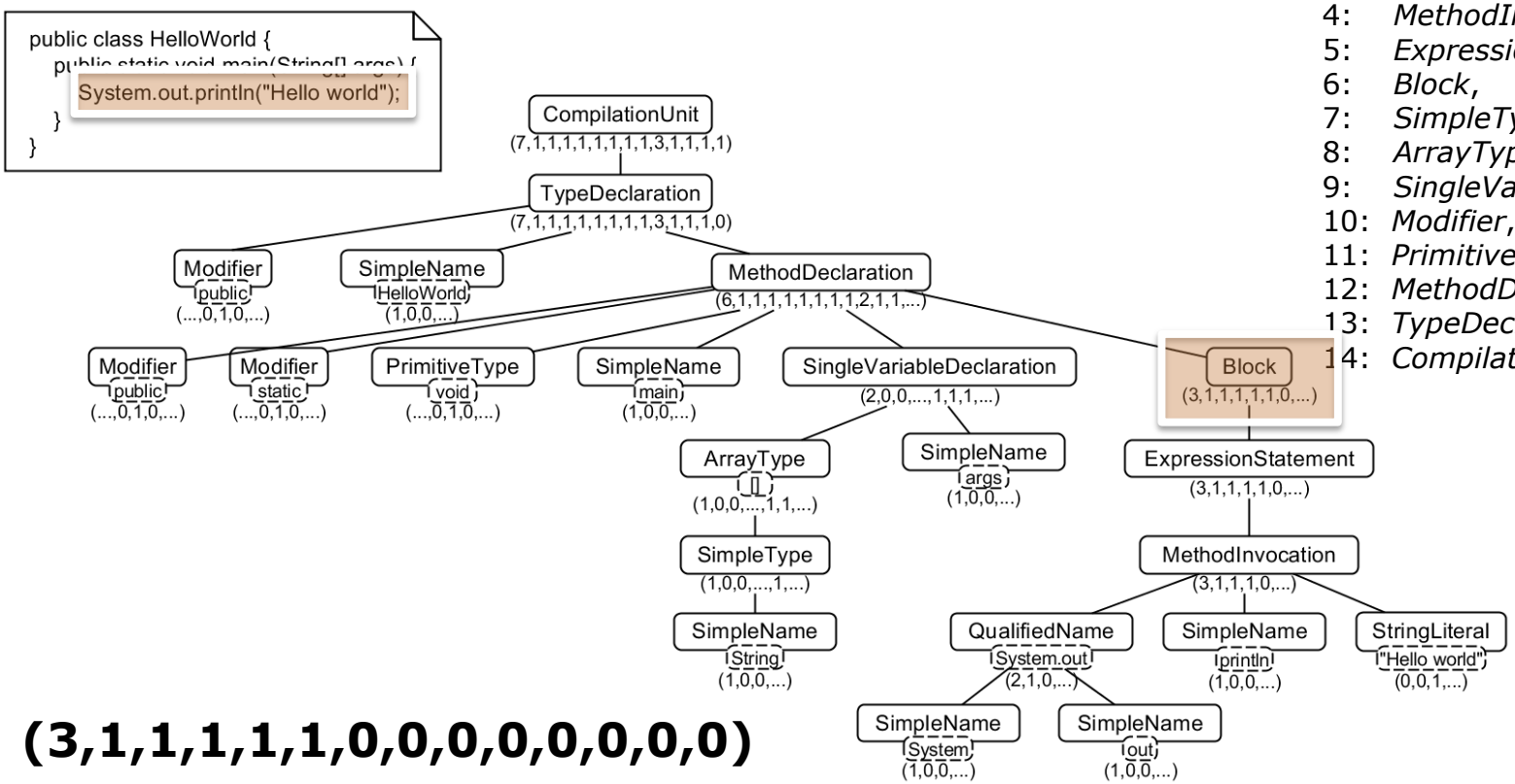


1: (*SimpleName*,
2: *QualifiedName*,
3: *StringLiteral*,
4: *MethodInvocation*,
5: *ExpressionStatement*,
6: *Block*,
7: *SimpleType*,
8: *ArrayType*,
9: *SingleVariableDeclaration*,
10: *Modifier*,
11: *PrimitiveType*,
12: *MethodDeclaration*,
13: *TypeDeclaration*,
14: *CompilationUnit*)

Deckard

- Vector generation

1: (*SimpleName*,
2: *QualifiedName*,
3: *StringLiteral*,
4: *MethodInvocation*,
5: *ExpressionStatement*,
6: *Block*,
7: *SimpleType*,
8: *ArrayType*,
9: *SingleVariableDeclaration*,
10: *Modifier*,
11: *PrimitiveType*,
12: *MethodDeclaration*,
13: *TypeDeclaration*,
14: *CompilationUnit*)

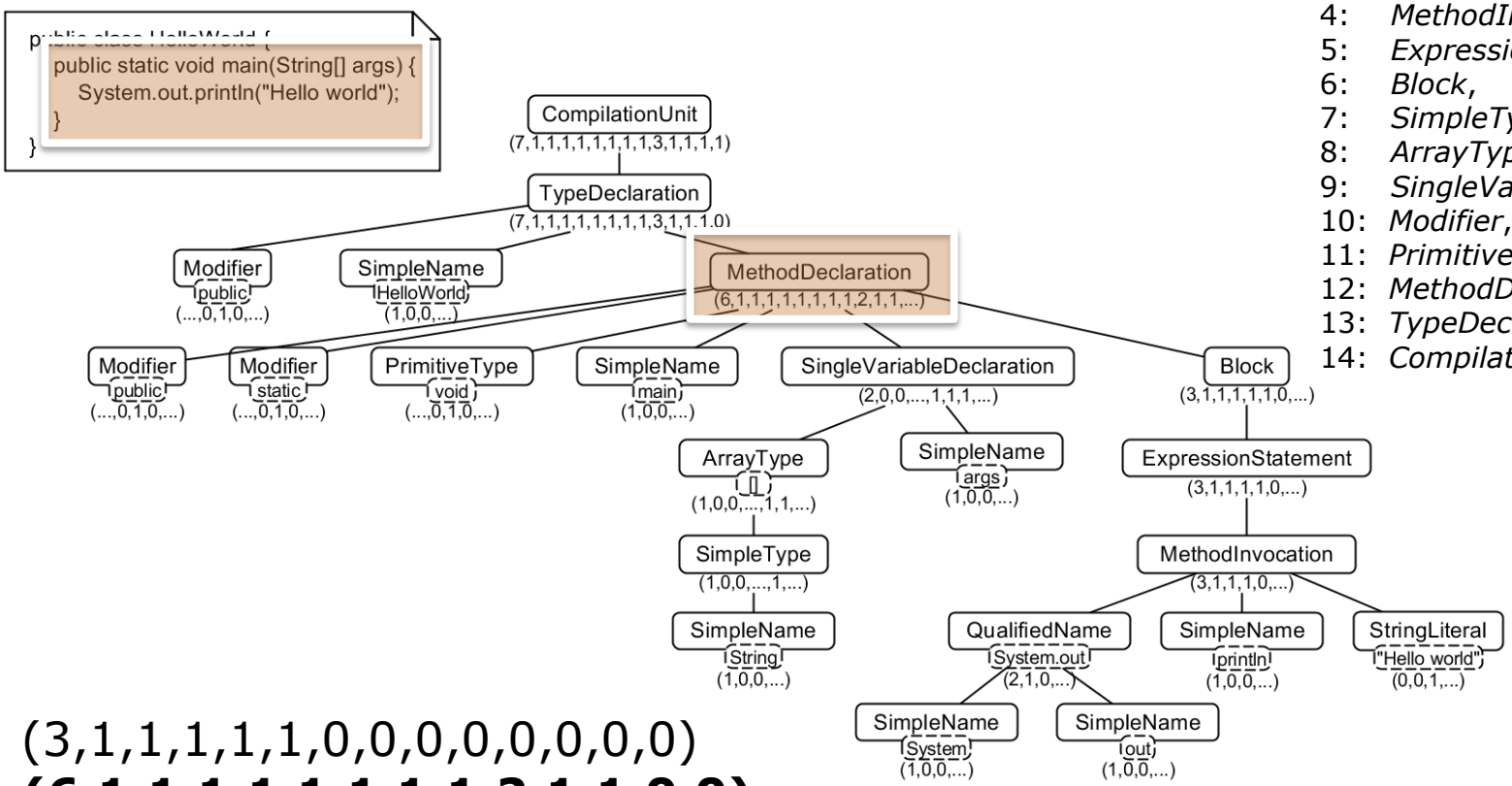


(3,1,1,1,1,1,0,0,0,0,0,0,0,0)

Deckard

- Vector generation

1: (*SimpleName*,
2: *QualifiedName*,
3: *StringLiteral*,
4: *MethodInvocation*,
5: *ExpressionStatement*,
6: *Block*,
7: *SimpleType*,
8: *ArrayType*,
9: *SingleVariableDeclaration*,
10: *Modifier*,
11: *PrimitiveType*,
12: *MethodDeclaration*,
13: *TypeDeclaration*,
14: *CompilationUnit*)

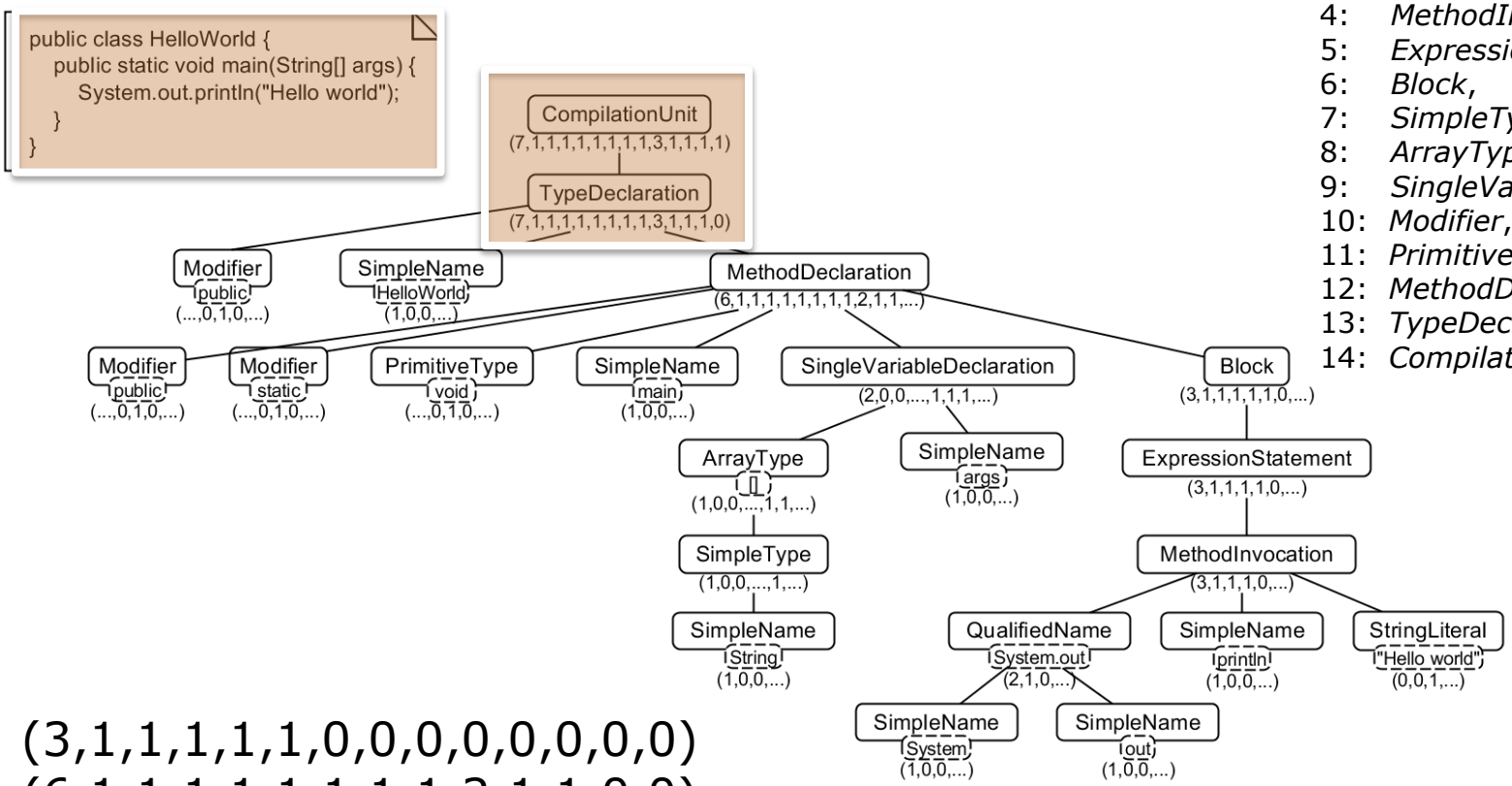


(3,1,1,1,1,1,0,0,0,0,0,0,0,0)
(6,1,1,1,1,1,1,1,1,2,1,1,0,0)

Deckard

- Vector generation

1: (*SimpleName*,
2: *QualifiedName*,
3: *StringLiteral*,
4: *MethodInvocation*,
5: *ExpressionStatement*,
6: *Block*,
7: *SimpleType*,
8: *ArrayType*,
9: *SingleVariableDeclaration*,
10: *Modifier*,
11: *PrimitiveType*,
12: *MethodDeclaration*,
13: *TypeDeclaration*,
14: *CompilationUnit*)



(3,1,1,1,1,1,0,0,0,0,0,0,0,0)
(6,1,1,1,1,1,1,1,1,2,1,1,0,0)
(7,1,1,1,1,1,1,1,1,3,1,1,1,0)
(7,1,1,1,1,1,1,1,1,2,1,1,1,1)

	Greedy strategy	Variance-based
1	Identifier	Identifier
2	ID_TK	ID_TK
3	Unary expression	Unary expression
4	Multiplicative expression	Multiplicative expression
5	Additive expression	Additive expression
6	Shift expression	Relational expression
7	Relational expression	Shift expression
8	Equality expression	Equality expression
9	Conditional expression	Conditional expression
10	Assignment expression	Assignment expression