LARA Tutorial

6. Java-to-Java Weaving

Tiago Carvalho, Pedro Pinto, João Bispo, Ricardo Nobre, Luís Reis, and João M.P. Cardoso

University of Porto, FEUP, Porto, Portugal

April 20th, 2016

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

ANTAREX 10^{18}

SPeCS
Special-Purpose Computing Systems, languages and tools

Objectives

- Demonstrate LARA in the context of Java
- Report static information
- Capture and report runtime information
- Transform code to allow adaptation

Some Information

- **Kadabra**
  - LARA + Spoon*
  - Ongoing Work!
  - http://specs.fe.up.pt/tools/kadabra/

- Example Used
  - Median smooth

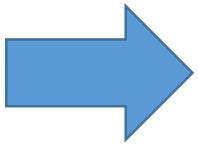
*Spoon library: http://spoon.gforge.inria.fr/

6.1 Static Code Report

- Print static information regarding
  - No. Of : files, **classes**, **methods** and calls
  - Loop and call information

New joinpoints: class, interface
Join point rename: function → method

```
select file.class.method.loop end
apply
  num_loops++;
  if($loop.type == 'for') {
   for_loops++;
   if($loop.isInnermost) {
     num_loops_innermost++;
     for_innermost_loops++;
   }
  }
  ...
end
```



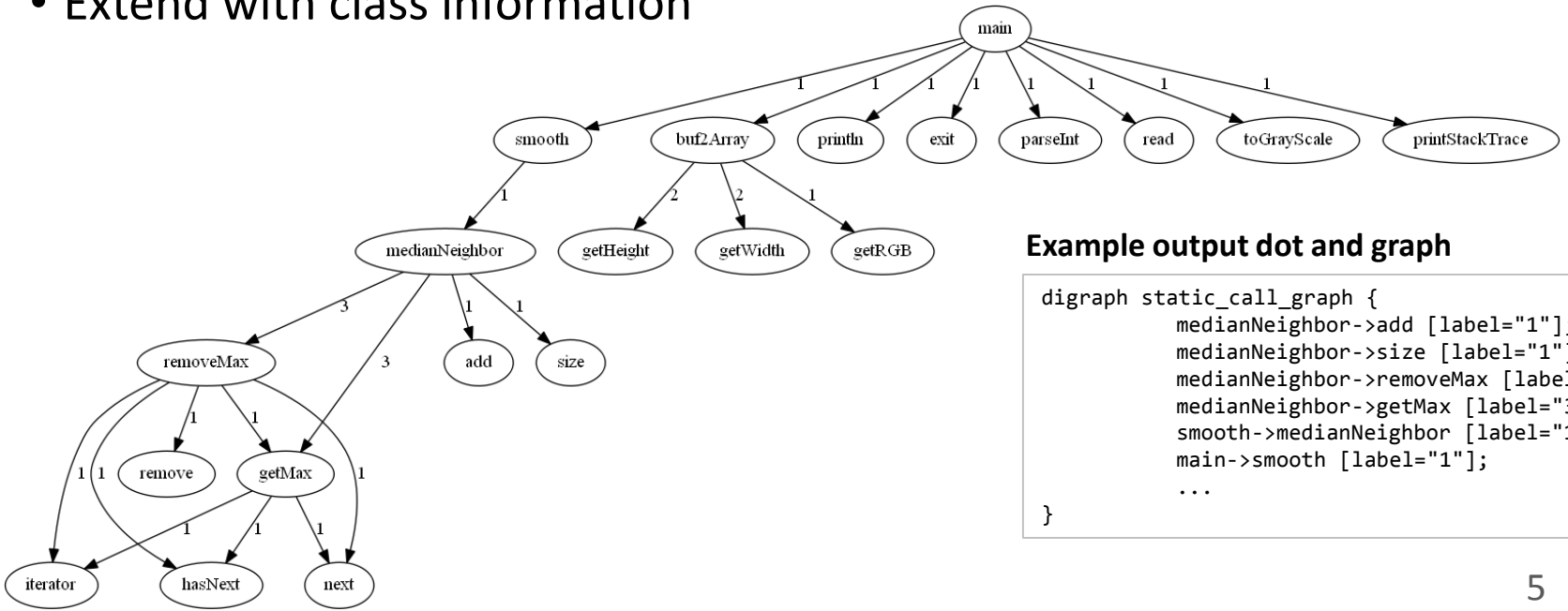
Example output:

```
-== [ Loop Information ] ==-

==============================
Type        Total    Innermost
------------------------------
For         6        6
While       0        2
Do-While    0        0
Foreach     7        0
------------------------------
Total       13       8
==============================
```

6.2 Static Call Graph - Goals

- Print a static call graph in dot format
- Extend with class information



Example output dot and graph

```
digraph static_call_graph {
            medianNeighbor->add [label="1"];
            medianNeighbor->size [label="1"];
            medianNeighbor->removeMax [label="3"];
            medianNeighbor->getMax [label="3"];
            smooth->medianNeighbor [label="1"];
            main->smooth [label="1"];
            ...
}
```

6.2 Static Call Graph - Strategy

- Select pairs method→call
- Count occurences of each pair
- Iterate LaraObject and print

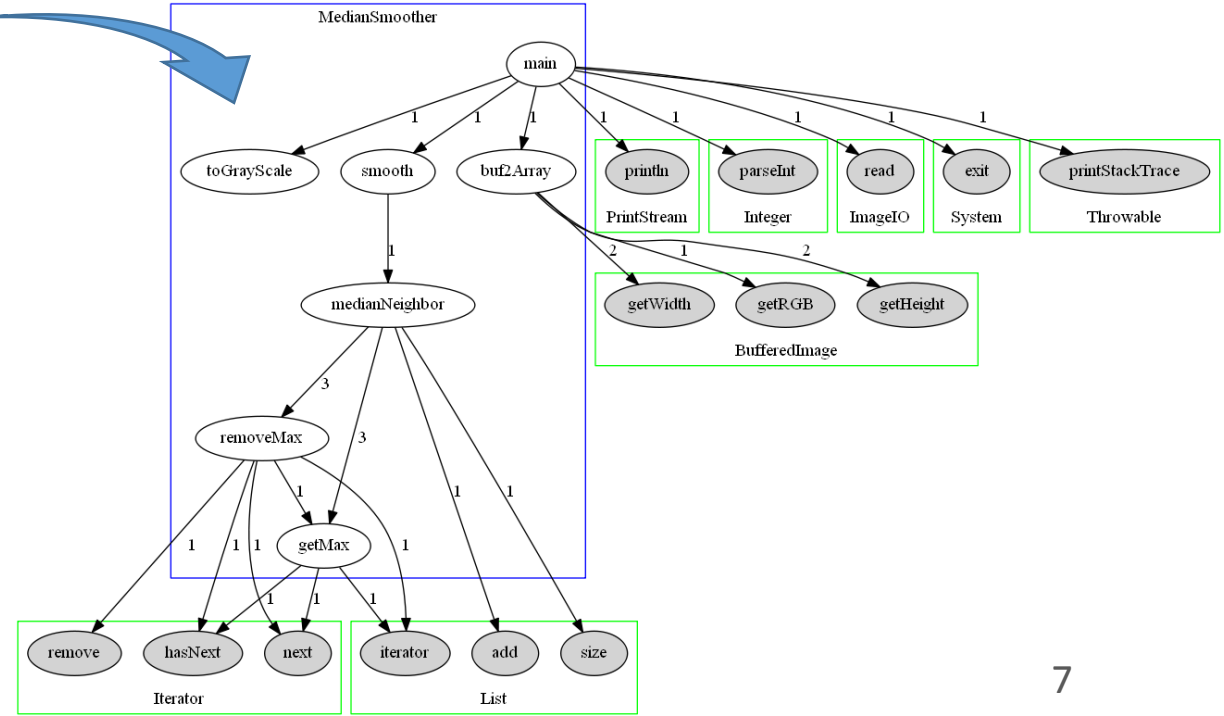
```
var call_graph = new LaraObject();
select class.method.call end
apply
  call_graph.inc($method.name, $call.name);
end
```

```
println('digraph static_call_graph {\n');
for (f in call_graph) {
 for (c in call_graph[f]) {
   println('\t' + f + '->' + c +
     ' [label="' + call_graph[f][c] + '"];');
 }
}
println('}');
```

6.2.1 Extended Static Call Graph - Strategy

- Add class information as clusters

```
subgraph cluster_MedianSmoother {
   label="MedianSmoother";
   color=blue;
   getMax
   medianNeighbor
   buf2Array
   smooth
   toGrayScale
   removeMax
   main
}
```



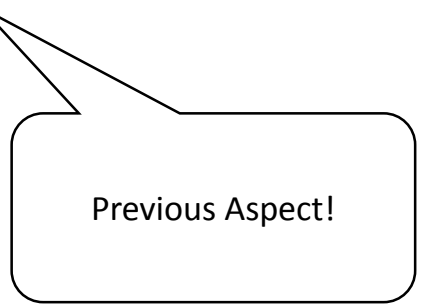
6.3 Dynamic Call Graph (DCG) – Goals

- Capture a call graph of an execution
- Generate dot code to represent the graph
- Extend with class information

6.3 Dynamic Call Graph (DCG) – Strategy

- Gather class→method information

```
call GetClasses();
```



6.3 Dynamic Call Graph (DCG) – Strategy

- Gather class→method information

- Insert a Monitor
  - Create new class
  - Add field in `$class`

    ```
    CountingMonitorList monitor;
    ```

- Add graph printer method

```
$class.exec newMethod(...,'void','printCallGraph',[]);
```

```
select class{name~=targetClass} end
apply
  call CountingMonitorList($class);
end
```

```
public class CountingMonitorList {
    private int[] counter;
    public CountingMonitorList(int size) {
        counter = new int[size];
    }
    public int get(int id) {
        return counter[id];
    }

    public void increment(int id) {
        counter[id]++;
    }
    ...
}
```

6.3 Dynamic Call Graph (DCG) – Strategy

- Gather class→method information

- Insert a Monitor
  - Create new class
  - Add field in `$class`
  - Add graph printer method

- Select method → call pairs
  - Define id for each pair
  - Use id to increment before calls

```
select method.call end
 apply
  var id = tuple.getId($method.name, $call.name);
  var incrementCode = monitor.increment(id);
  insert before '[[incrementCode]]';
 end
end
```

```
monitor.increment(5);
removeMax(values);
```

6.3 Dynamic Call Graph (DCG) – Strategy

- Gather class→method information
- Insert a Monitor
  - Create new class
  - Add field in `$class`
  - Add graph printer method
- Select method → call pairs
  - Define id for each pair
  - Use id to increment before calls
- Insert code in printer to output dot

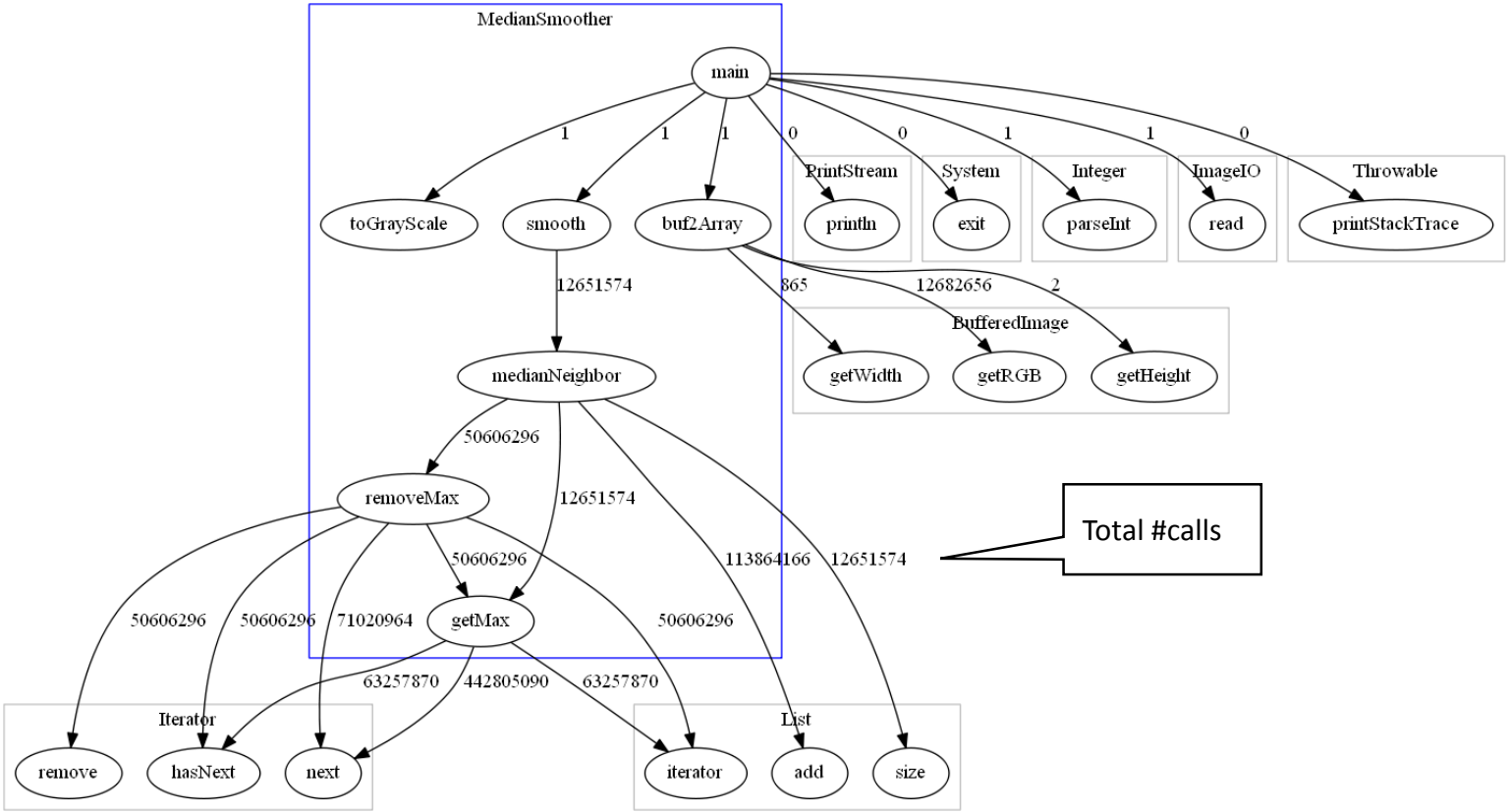
```
select class.method{'printCallGraph'}.body end
 apply
   call printer:GenerateDCGPrinter(monitor, pair,
                                   classes, apis);
   insert replace '[[printer.code]]';
 end
```

```
aspectdef GenerateDCGPrinter
  input monitor, obj, methods, apis  end
  output code end
  code  = 'int value;\n'
  code += '\tSystem.out.println("digraph DCG {';

  for(var c in methods) {

    code += '\tSystem.out.println(...);';
    ...
end
```

6.3 Dynamic Call Graph (DCG) – Output



6.4 Method Call Adaptation – Goals

- Several algorithms to calculate median
  - Removing half max. values (current approach)
  - Counting sort
  - Sorting Network
- Replace median method call
- Select best algorithm based on arguments

```
int[][] smooth(int[][] input, int kernelSize,...) {
 int limit = kernelSize / 2;
 int[][] outputArrays = new int[height][width];
 for (int i = limit ; i < (height - limit) ; ++i) {
    for (int j = limit ; j < (width - limit) ; ++j) {
     outputArrays[i][j] = medianNeighbor(input, kernelSize);
    }
 }
 return outputArrays;
}
```

6.4 Method Call Adaptation – Strategy

- Extract a functional interface (FI)*

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static interface IMedianNeighbor {
    public int medianNeighbor(int[][] input, int kernelSize, int width,
                              int height, int y, int x);
}
```

```
call ExtractFuncInterface('MedianSmoother', 'medianNeighbor');
```

* Functional Interfaces: https://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html
https://www.oreilly.com/learning/java-8-functional-interfaces

6.4 Method Call Adaptation – Strategy (2)

- Create a new field of type FI

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static interface IMedianNeighbor {
    public int medianNeighbor(int[][] input, int kernelSize, int width,
                              int height, int y, int x);
}

static IMedianNeighbor medianNeighbor = MedianSmoother::medianNeighbor;
```

```
$class.exec newField(['static'], 'IMedianNeighbor', 'medianNeighbor', MedianSmoother::medianNeighbor);
```

6.4 Method Call Adaptation – Strategy (3)

- Replace call with field invocation

```
$call.def target = 'medianNeighbor';
```

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static int[][] smoo          input, int kernelSize, int width, int height)
    int limit = kernelSize /
    int[][] outputArrays = new          eight][width];
    for (int i = limit ; i < (he        t - limit) ; ++i) {
        for (int j = limit ; j <        idth - limit) ; ++j) {
            outputArrays[i][j] = medianNeighbor.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}


public static interface IMedianNeighbor {
    public int medianNeighbor(int[][] input, int kernelSize, int width,
                              int height, int y, int x);
}

static IMedianNeighbor medianNeighbor = MedianSmoother::medianNeighbor;
```

6.4.1 Switch Call – Strategy

- Insert a switch to select between versions
- Insert versions
  - Sorting network: kernelSize == 3
  - Counting sort: kernelSize == 5 || 7

```
codedef SwitchVersion(fieldName, className, defaultMethod) %{
  switch(kernelSize){
    case 3:
      [[fieldName]] = [[className]]::sortingNetwork; break;
    case 5:
    case 7:
      [[fieldName]] = [[className]]::countingSort; break;
    default: [[fieldName]] = [[className]]::[[defaultMethod]]; break;
  }}%
end
```

6.4.1 Switch Call – Diff

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = MedianSmoother.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}
```

```
public static int[][] smooth(int[][] input, int kernelSize, int width, int height)
    switch(kernelSize){
            case 3: medianNeighbor = MedianSmoother::sortingNetwork; break;
            case 5:
            case 7:
                    medianNeighbor = MedianSmoother::countingSort; break;
            default: medianNeighbor = MedianSmoother::medianNeighbor; break;
    };
    int limit = kernelSize / 2;
    int[][] outputArrays = new int[height][width];
    for (int i = limit ; i < (height - limit) ; ++i) {
        for (int j = limit ; j < (width - limit) ; ++j) {
            outputArrays[i][j] = medianNeighbor.medianNeighbor(input, kernelSize,
                                                               width, height, i, j);
        }
    }
    return outputArrays;
}

public static int medianNeighbor(int[][] input, int kernelSize, int width,
                                 int height, int y, int x) {
    ...
}


public static interface IMedianNeighbor {
    public int medianNeighbor(int[][] input, int kernelSize, int width,
                              int height, int y, int x);
}

static IMedianNeighbor medianNeighbor = MedianSmoother::medianNeighbor;
```

```
insert before SwitchVersion('medianNeighbor', //field
                            'MedianSmoother', //class
                            'medianNeighbor');//default
```

6.4.2 Map Call – Strategy

- New class mapping kernelSize → method
- Add default versions
- Insert getter to the best version

Can add/reassign new versions at runtime!

```
call methodCaller: NewMappingClass(interface, baseName, 'int');

//Add some default versions
var versionsCode = 'put(3, MedianSmoother::sortingNetwork);';
...
$mapClass.exec insertStatic(versionsCode);

var getBestVersionCode = methodCaller.get('kernelSize', defaultMethod);
//at the beginning of smooth assign the best version
$first_stmt.insert before '[[fieldName]] = [[getBestVersionCode]]';
```

6.4.2 Map Call – Mapping class

```
call methodCaller: NewMappingClass(interface, baseName, 'int');
```

```
1. public class MedianNeighborCaller {
2.     final public static Map<Integer, IMedianNeighbor> map= new ConcurrentHashMap<>();
3.
4.     public static void put(int key, IMedianNeighbor IMedianNeighbor) {
5.         map.put(key,IMedianNeighbor);
6.     }
7.
8.     public static boolean contains(int key) {
9.         return map.containsValue(key);
10.    }
11.
12.    public static IMedianNeighbor get(int key, IMedianNeighbor defaultMethod) {
13.        return map.getOrDefault(key, defaultMethod);
14.    }
15. }
```

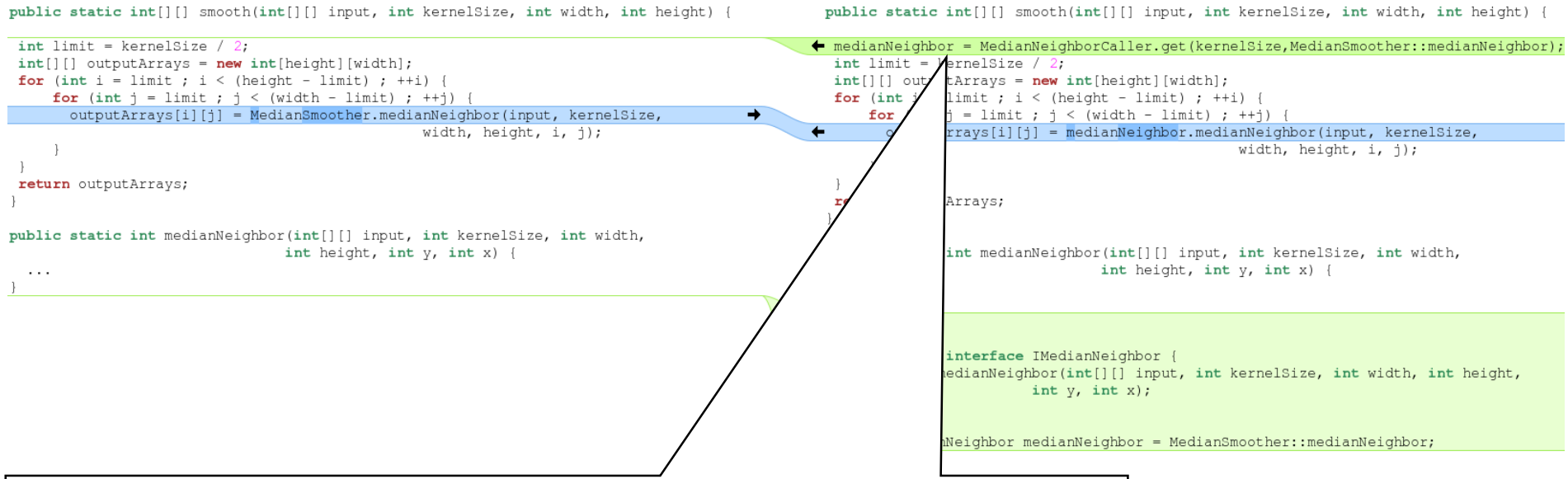
6.4.2 Map Call – Mapping class

```
var versionsCode = 'put(3, MedianSmoother::sortingNetwork);';
...
$mapClass.exec insertStatic(versionsCode);
```

```
1. public class MedianNeighborCaller {
2.     static {
3.         put(3, MedianSmoother::sortingNetwork);
4.         put(5, MedianSmoother::countingSort);
5.         put(7, MedianSmoother::countingSort);
6.     }
7.
8.     final public static Map<Integer, IMedianNeighbor> map= new ConcurrentHashMap<>();
9.
10.    public static void put(int key, IMedianNeighbor IMedianNeighbor) {
11.        map.put(key,IMedianNeighbor);
12.    }
13.
14.    public static IMedianNeighbor get(int key, IMedianNeighbor defaultMethod) {
15.        return map.getOrDefault(key, defaultMethod);
16.    }
17. }
```

Default Versions

6.4.2 Map Call – Diff




```
var getBestVersionCode = methodCaller.get('kernelSize', 'MedianSmoother::medianNeighbor');
$first_stmt.insert before '[[fieldName]] = [[getBestVersionCode]]';
```

Takeaway Points

- Some aspects can be reused between tools
  - Minor changes: function→method
- Code can be transformed to comprise adaptation
- Provides mechanisms for runtime adaptability
  - Through static insertion/transformation