Inner State Capture with AspectJ

Prepared by Onur YÜKSEL

Summer 2011

1. Introduction to AspectJ

AspectJ is an aspect-oriented extension for the Java programming language. AspectJ has become a widely-used tool for aspect oriented programming with its simplicity and usability. All valid Java programs are also valid AspectJ programs, but AspectJ is for allowing programmers to define special constructs called aspects.

An **AspectJ aspect** is a crosscutting type consisting of advice on pointcuts lexical introduction of behavior into other types like classes, aspects can have ;

- internal state and behavior,
- can extend other aspects and classes,
- and can implement interfaces

A **Join point** is a well-defined location at a point in the execution of a Program.
(e.g. the execution of the method public void `A.foo(int)`)

A **Pointcut** is a set of join points all method calls to class B within class A all mutations of fields of class A outside of A's subclasses.

An **Advice** is a piece of code designed to run automatically at all join points in a particular pointcut can be marked as before , after , or around (in place of) the join points in the pointcut.

2. AspectJ Functionality in Inner State Capture

2.1 Pointcuts

Since it is needed to collect all inputs and outputs of all functions over execution time as they run, a join point that defines all function executions is defined :

- `pointcut functionExecuted() : execution(* *(..));`

(first asterisk is for selecting all types, second asterisk is for selecting all names, two dots are for selecting all parameters)

But in this form, this pointcut also selects functions that run in aspect j side.So it is needed to seperate them as follows :

- ```
  pointcut functionExecuted() : execution(* *(..)) && !execution(void
  main(..))&& !execution(void handleArrays(..))&& !execution(void
  hash(..))&& !execution(void handleObjects(..))&& !execution(String
  convertToHex(..))&& !execution(void addToInnerState(..))&&
  !execution(void addition(..))&& !execution(void writeInnerState(..));
  ```

Other pointcut is for reaching fields of objects whenever they are set :

- ```
  pointcut setField() : set(* *) && !set(String innerState);
  ```

It is needed to seperate innerState field from this pointcut otherwise it enters to infinite loop.

There are two additional pointcuts in this project.One is for reaching all function return values.To reach them, function calls must be defined as follows :

- ```
  pointcut  functionCall(): execution(* *(..))&& !execution(void
  main(..))&& !execution(void INNERSTATEKUhandleArrays(..))&&
  !execution(void INNERSTATEKUhash(..))&& !execution(void
  INNERSTATEKUhandleObjects(..))&& !execution(String
  INNERSTATEKUconvertToHex(..))&& !execution(void
  INNERSTATEKUaddToInnerState(..))&& !execution(void
  INNERSTATEKUaddition(..))&& !execution(void
  INNERSTATEKUwriteInnerState(..));
  ```

  and :

- ```
  pointcut mainExecution() : execution(void main(..));
  ```

2.2 Advices

To reach functions inputs following advice is defined :

- ```
  before() : functionExecuted(){…}
  ```

This advice first, captures functions name, adds it to inner state.

```
(String functionName = thisJoinPoint.getSignature().getName();)
```

Then retrieves all inputs in an object array.

```
(Object[] args = thisJoinPoint.getArgs();)
```

This object array then passed to addToInnerState function which is discussed later.

Other advice is for reaching a functions output value :

- ```
  after() returning(Object returnVal):functionCall(){…}
  ```

Here, the type of the returnVal is object but it captures all kinds of return values since all values are converted to objects in Java.It passes returnVal to addition function directly.The content of the function is discussed later.

- `after(): setField(){…}`

This advice first retrieves assigned values in an object array.

`(Object[] args = thisJoinPoint.getArgs();)`

This object array then passed to addToInnerState function.

Final advice is for writing inner state after main execution.

- `after():mainExecution(){…}`

2.3 Functions used in advices

In order to support readability addition to inner state is seperated into a function called addToInnerState.This function simply checks all elements in object list, calls addition function -if it is not null-, then checks for the length constant for hashing.

```
private void INNERSTATEKUaddToInnerState(Object [] argumentList)
	{
		for(int i=0;i<argumentList.length;i++) //for all arguments that
passed to a function
		{
			if(argumentList[i]!=null)
			{
				INNERSTATEKUaddition(argumentList[i]);//add to inner
state
			}
			else //if null parameter passed
			{
				innerState=innerState+"null";
			}

			if(innerState!=null)
			{
				if(innerState.length()>lengthConstant)
				{
					INNERSTATEKUhash();
				}
			}
			}
```

Addition function is also seperated to add single parameters and return values to inner state.If single parameter is array,then it must be handled in a different way so handleArrays function is called.Else we check for '@' character in oreder to prevent containing java's hash code in inner state, if so,handleObjects function is called. Addition function as follows :

```
private void INNERSTATEKUaddition(Object singleParameterOrReturnValue)//Addition
to inner state
	{

	if(singleParameterOrReturnValue.getClass().isArray())//argumentList[i] is an
array parameter
		{
			Object arr = new Object();
			arr=singleParameterOrReturnValue;
			INNERSTATEKUhandleArrays(arr);
		}
		else
		{

	if(singleParameterOrReturnValue.toString().contains("@"))//argumentList[i]
is an object parameter
			{
				String
hashCodeOfObject=Integer.toHexString(singleParameterOrReturnValue.hashCode());//ha
sh code of the object
				String oldVal=singleParameterOrReturnValue.toString();


	if(singleParameterOrReturnValue.toString().contains(hashCodeOfObject))//cont
ains hash code of the object (java's toString method for objects has run)
				{
					innerState=innerState+oldVal.substring(0,
oldVal.indexOf("@"));//remove hash value part;
				}
				else
				{
					if(oldVal.contains(","))//if a list type object
(array list,hash table,stack,queue etc.)
					{

	oldVal=singleParameterOrReturnValue.toString();
						INNERSTATEKUhandleObjects(oldVal);
					}
					else
					{

	innerState=innerState+singleParameterOrReturnValue;
					}
				}


			}
			else// argumentList[i] is a primitive type
			{
				innerState=innerState+singleParameterOrReturnValue;
			}
```

```
		}

		}
```

HandleArrays function is for reaching the contents of arrays by casting them to their original types :

```
private void INNERSTATEKUhandleArrays(Object arr)
	{
		if(arr instanceof int[])  //choose type
		{
			int[] list=(int[])arr;
			Object[] objList=new Object[list.length];
			for(int i=0;i<objList.length;i++)
			{
				objList[i]=new Object();
				objList[i]=list[i];//copy to object list in order to use
Arrays.deepToString
			}
			innerState=innerState+(Arrays.deepToString(objList));//add
array to string (deeper to its elements)
		}
		else if(arr instanceof char[])
		{
			char[] list=(char[])arr;
			Object[] objList=new Object[list.length];
			for(int i=0;i<objList.length;i++)
			{
				objList[i]=new Object();
				objList[i]=list[i];//copy to object list in order to use
Arrays.deepToString
			}
			innerState=innerState+(Arrays.deepToString(objList));//add
array to string (deeper to its elements)

		}
		else if(arr instanceof byte[])
		{
			…
		}
		else if(arr instanceof short[])
		{
			…
		}
		else if(arr instanceof long[])
		{
			…
		}
		else if(arr instanceof float[])
		{
			…
		}
		else if(arr instanceof double[])
		{
			…
		}
		else if(arr instanceof boolean[])
		{
```

```
                …
        }
        else if(arr instanceof String[])
        {
                …
        }

        if(innerState!=null)
        {
                if(innerState.length()>lengthConstant)//if bigger than length
constant hash
                {
                        INNERSTATEKUhash();
                }
        }

        }
```

Finally, it checks for length constant in case of hashing required.

Handleobjects function,as comments in the function emphasized,removing java's possible hash function values attached when toString method missing in that object.

```
private void INNERSTATEKUhandleObjects(String oldVal)
      {
              String newVal="";
              String temp="";
              String check="";

                      String[] list=oldVal.split(",");
                      for(int k=0;k<list.length;k++)
                      {
                              if(!list[k].contains("@"))//if not contains java's hash
function value
                              {
                                      newVal=newVal+list[k];
                              }
                              else
                              {
                                      temp=list[k];
                                      temp=temp.trim();
                                      if(temp.charAt(0)!='@')
                                      {
                                              int previousIndexOfAt=temp.indexOf('@')-1;
                                              int nextIndexOfAt=previousIndexOfAt+2;


      if(temp.charAt(previousIndexOfAt)!='['&&temp.charAt(previousIndexOfAt)!='
'&&temp.charAt(previousIndexOfAt)!=',')
                                              {

      while(nextIndexOfAt<temp.length()&&temp.charAt(nextIndexOfAt)!=']'&&temp.cha
rAt(nextIndexOfAt)!=' '&&temp.charAt(nextIndexOfAt)!=',')
                                                      {
```

```
	check=check+temp.charAt(nextIndexOfAt);
                                                nextIndexOfAt++;
                                        }

                                        if(!check.matches[a-fA-f0-9][a-fA-f0-
9][a-fA-f0-9][a-fA-f0-9][a-fA-f0-9][a-fA-f0-9][a-zA-z0-9][a-fA-f0-9]"))
                                        {
                                                newVal=newVal+list[k];
                                                check="";
                                        }
                                }

                        }
                        else
                        {
                                newVal=newVal+list[k];
                        }

                }

        }
        innerState=innerState+newVal;
	}
```

Note that if string oldVal contains ',' it means that it is an array type object.So objects not containing toString method can occur in hash tables,stacks etc.This condition is for handling the situation.

Example values :

If original string is `[abcde, abcde@, abcde@yahoo.com, sortingPackage.Book@2586db54, @abcdeTwitter, a@bcde@yahoo.com@]` (when toString of an object is called)

This method will produce a string to add to the inner state is :

`[abcde abcde@ abcde@yahoo.com @abcdeTwitter a@bcde@yahoo.com@]`

Three functions are used for hashing mechanism.First one is called hash.It is the main hash function,it calls secureHashAlgorithm surrounded with exception constructors.

```
private void INNERSTATEKUhash()
	{
		try {
                        INNERSTATEKUsecureHashAlgorithm();
                } catch (UnsupportedEncodingException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (NoSuchAlgorithmException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

            }
```

SecureHashAlgorithm is for typical SHA-1 hashing, as follows :

```
private void INNERSTATEKUsecureHashAlgorithm() throws
UnsupportedEncodingException, NoSuchAlgorithmException
      {
            byte[] theTextToDigestAsBytes = (innerState).getBytes("iso-8859-
1");//Can also be "UTF8"
            MessageDigest md = MessageDigest.getInstance(hashAlg);
        md.update( theTextToDigestAsBytes, 0, innerState.length());
        byte[] digest = md.digest();
        String newVal = "";
        newVal=INNERSTATEKUconvertToHex(digest);
        innerState=newVal;
    }
```

convertToHex function is simply converts byte to hex.

```
private static String INNERSTATEKUconvertToHex (byte[] data) {

            StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9))
                    buf.append((char) ('0' + halfbyte));
                else
                    buf.append((char) ('a' + (halfbyte - 10)));
                halfbyte = data[i] & 0x0F;
            } while(two_halfs++ < 1);
        }
        return buf.toString();
    }
```

At the end of main function writeInnerState function is called:

```
private void INNERSTATEKUwriteInnerState ()
      {
            if(innerState.contains("null"))//In the end,after hashed null can be
seen at the end of inner state (unknown reason)
            {
                  innerState=innerState.replace("null", "");
            }
            System.out.println(innerState);
          }
```

3. Evaluation

Different types of tests are used in order to test the functionality and the correctness.

3.1 Change Test

- Selection sort algorithm :

```
public static int[] selectionSort(int[] arr)
{
	for(int i=0;i<arr.length;i++)
	{
		int min=i;
		for(int index=min;index<arr.length;index++)
			if(arr[index]<arr[min])
				min=index;
		int temp=arr[i];
		arr[i]=arr[min];
		arr[min]=temp;
	}
	return arr;
}
```

```
Final hash value : d480 6183 2997 bb6f 961e 398c f657 5119 f2ac 4ec2
```

- Manipulated Code (with same inputs):

```
public static int[] selectionSort(int[] arr)
{
	for(int i=0;i<arr.length;i++)
	{
		int min=i;
		for(int index=min;index<arr.length;index++)
			if(arr[index]>arr[min])
				min=index;
		int temp=arr[i];
		arr[i]=arr[min];
		arr[min]=temp;
	}
	return arr;
}
```

```
Final hash value : 8fc3 5ec1 63e2 816d ca20 2d7b 9e51 dfb4 bcf0 5a4f
```

- Factorial Algorithm :

```
public static int factorial(int n)
{
	if(n>1)
```

```
            return n*factorial(n-1);
            else
                  return 1;

      }
```

```
Final hash value : 6041 bb83 d8f1 56a2 ae5a 7991 e72c 4e33 dcf6 fa24
```

- Manipulated Code (with same inputs):

```
      public static int factorial(int n)
      {
            if(n<1)
            return n*factorial(n-1);
            else
                  return 1;

      }
```

```
Final hash value : a5c8 c6a3 c5fb f1ff 6cd7 dc66 3b33 1981 2e53 3521
```

- Boolean Calculations :

```
      public static boolean and(boolean x1,boolean x2)
      {
            if(x1==true & x2==true)
                  return true;
            else
                  return false;
      }
      public static boolean or(boolean x1,boolean x2)
      {
            if(x1==true | x2==true)
                  return true;
            else
                  return false;
      }

      public static boolean xor(boolean x1,boolean x2)
      {
            if(x1==true)
                  return !x2;
            else
                  return x2;
      }

boolean b1=true;
boolean b2=false;
boolean b3=true;
boolean b4=false;

b1=and(xor(or(and(b1,b2),b3),b4),b1);
```

```
Final hash value : 88aa d95a 49db 7f8f 196e a62e 5935 b081 87ec c78c
```

- Manipulated Code (with same inputs):

```
public static boolean and(boolean x1,boolean x2)
{
            return false;
}
public static boolean or(boolean x1,boolean x2)

            return false;
}

public static boolean xor(boolean x1,boolean x2)
{
            return x2;
}
```

```
Final hash value : 21c5 c260 7124 eb79 d560 a1f0 431d b77a d0e2 609b
```

Note : If contractors make the functions return random boolean values,and if the boolean functions will result according to the same order as the true copy, hash value will be the same as the true one.

As seen obviously, if contractors try to spoil the algorithm or try to send tricky guesses hash value make them be caught by the boss.

3.2 Different Source Code Test

This test was performed to see results of the inner state system on different source codes.Re-generatable inputs used for future checks.

- Shell Sort

```
Input:100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76
75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49
48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22
21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

```
Output:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Executed Code :

```
int maxSize = 100;
ShellSort arr = new ShellSort(maxSize);

for (int j = 0; j < maxSize; j++) {
long n = maxSize-j;
arr.insert(n);
}
```

```
arr.display();
arr.shellSort();
arr.display();
```

Final Hash Value :

```
7458 61c4 210b 1a04 7d31 ccd7 69b6 8876 c1d3 b3d2
```

Note : Look at ShellSort.java file for further information.

- Rational Number Operations

Executed Code :

```
Rational X=new Rational(700000, 69565);
Rational Y=new Rational(789845, 78510);
X.divide(Y);
```

Final Hash Value :

```
fec2 f9a1 9ed2 51da 8e0b 96b3 379b 972c bf9c b9f0
```

Note : Look at Rational.java and Super.java files for further information.

- Open Pit Mining

Executed Code :

```
OpenPitMining pit = new OpenPitMining();
System.out.println("I'm Memoization");
long x = System.currentTimeMillis();// Get System Time used for real time analysis
pit.solve(); // Solve

long y = System.currentTimeMillis();// Finish time

System.out.println("total time "+ (y-x));// Total time calculating
```

Note : Look at OpenPitMining.java and Test.txt files for further information.

Final Inner State : 1782 7a48 f4ce 5a40 7f94 7252 e28b e1ec 73b6 89e8

- Recursive Factorial Calculation

Executed Code :

```
factorial(1000);
```

```
Final Inner State : e537 576e c0a5 4939 c27c 429d 8288 57a8 ead1 5148
```

- Prime Number Detect

```
Input :1000

Output :False
```

Executed Code :

```
int checkVal=1000;
for(int theNum=1; theNum<=checkVal; theNum++)
{
isPrime=true;
for(int j=2; j<theNum; j++)
{
if((theNum%j)==0)
{
isPrime=false;
}
}}
```

```
Final Inner State : 7741 6dee c985 6e5a 89e1 84b5 417a 4aab 6b75 fa21
```

- Hash Table Operations

```
Input :Empty Hashtable

Output:Hashtable that has 1000 elements
```

Executed Code :

```
Hashtable htbl=new Hashtable();
htbl=hashTableOperation(htbl);

public static Hashtable hashTableOperation(Hashtable tbl)
{
for(int i=0;i<1000;i++)
{
tbl.put(i, "xxx");
}
return tbl;
}
```

Final Inner State : `7264 e676 4507 b507 94d9 2efe 369a 6f45 5a46 dfaa`

As shown, this inner state capturing system would work on different kinds of algorithms.

3.3 Speed Test

It is obvious that inner state capturing system would increase run time, but this test was performed to show tolerable limitations on run time increase.

Tha table below shows the results for run time increase of different algorithms with different length constant (hashing period).

Algorithm	Measured Time	Comments	
Prime Number Detect (lengthConstant=1000)	With AspectJ : 94,105 Without : 93,893	%0,226 increased	100,000 numbers detected if they are prime or not (numbers starting from 1,000,000 and increasing)
Prime Number Detect (lengthConstant=2000)	With AspectJ : 97,052 Without : 93,893	%3,364 increased	100,000 numbers detected if they are prime or not (numbers starting from 1,000,000 and increasing)
Prime Number Detect (lengthConstant=3000)	With AspectJ : 97,096 Without : 93,893	%3,411 increased	100,000 numbers detected if they are prime or not (numbers starting from 1,000,000 and increasing)
Prime Number Detect (lengthConstant=500)	With AspectJ : 94,102 Without : 93,893	%0,222 increased	100,000 numbers detected if they are prime or not (numbers starting from 1,000,000 and increasing)
Merge Sort (lengthConstant=1000)	With AspectJ : 35,053 Without :34,811	%0,695 increased	Array of int (Size: 500,000),sorted Randomly initialized and sorted 1000 times
Merge Sort (lengthConstant=2000)	With AspectJ : 34,896 Without :34,811	%0,244 increased	Array of int (Size: 500,000),sorted Randomly initialized and sorted 1000 times
Merge Sort (lengthConstant=3000)	With AspectJ : 35,110 Without :34,811	%0,859 increased	Array of int (Size: 500,000),sorted Randomly initialized and sorted 1000 times
Merge Sort (lengthConstant=500)	With AspectJ : 35,057 Without :34,811	%0,707 increased	Array of int (Size: 500,000),sorted Randomly initialized and sorted 1000 times

Pit Mining (lengthConstant=1000)	With AspectJ :55,220 Without :54,921	%0,544 increased	200 pits solved,input txt :25 kb
Pit Mining (lengthConstant=2000)	With AspectJ : 55,041 Without :54,921	%0,218 increased	200 pits solved,input txt :25 kb
Pit Mining (lengthConstant=3000)	With AspectJ : 55,253 Without :54,921	%0,604 increased	200 pits solved,input txt :25 kb
Pit Mining (lengthConstant=500)	With AspectJ : 55,121 Without :54,921	%0,364 increased	200 pits solved,input txt :25 kb

Obviously it is nearly impossible to pick a length constant that is optimal for all kinds of algorithms.But it is suggested that using a length constant 1000 would be tolerable.Also, if implemented,length constant can be a parameter of the system,decided by the boss according to algorithm itself.