Assignment-3

Coverage tool description: I am using IntelliJ idea Ultimate IDE, it provides a platform to perform code coverage. One can configure run as well as debug environments by specifying coverage options based on the requirements. By default IntelliJ comes with the code coverage plugin; one has to enable the plugin and choose the coverage data by selecting the file using “show code coverage data”. The plugin helps us to figure out the effectiveness and which parts of the code are executed when running the tests. It comes with two options: sampling and tracing. Sampling supports only line,method, class coverage whereas tracing which traces per test coverage supports branch coverage (which covers both conditional and unconditional branches) in addition to line,class, method coverages acting as a superset of decision coverage. All the coverages are expressed in percentage.

Types of coverage included in the IntelliJ code coverage plugin:

- Line coverage
- Method Coverage
- Branch Coverage
- Class Coverage

Source Listing

```
// Java program for implementation of Heap Sort
public class HeapSort {
    public void sort(int arr[])
    {
        int n =  arr.length;


        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is the size of heap
    void heapify(int arr[], int n, int i)
            largest = r;

        // If largest is not root
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

//    // Driver code
//    public static void main(String args[])
//    {
//        int arr[] = { 12, 11, 13, 5, 6, 7 };
//        int n = arr.length;
//
//        HeapSort ob = new HeapSort();
```

```
    {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest]
```

```
//        ob.sort(arr);
//
//        System.out.println("Sorted array is");
//        printArray(arr);
//    }
}
```

Test Cases used:

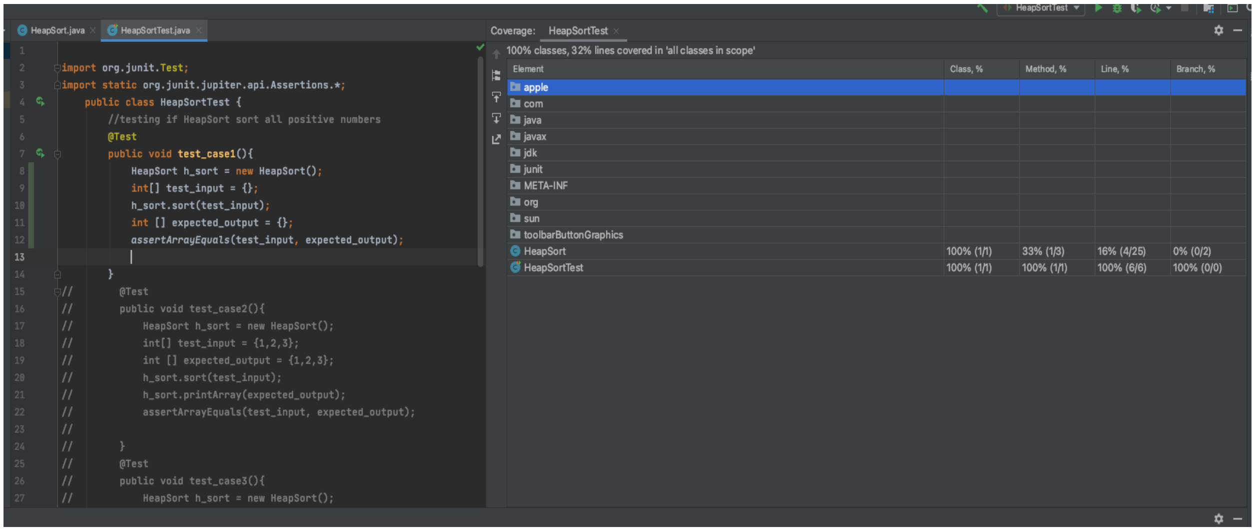
```

import org.junit.Test;
import static org.junit.jupiter.api.Assertions.*;
    public class HeapSortTest {
        //testing if HeapSort sort all positive
numbers
        @Test
        public void test_case1(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {};
            h_sort.sort(test_input);
            int [] expected_output = {};
            assertArrayEquals(test_input,
expected_output);

        }
        @Test
        public void test_case2(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {8};
            h_sort.sort(test_input);
            int [] expected_output = {8};
            assertArrayEquals(test_input,
expected_output);

        }
        @Test
        public void test_case3(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {1,2,3};
            int [] expected_output = {1,2,3};
            h_sort.sort(test_input);
            h_sort.printArray(expected_output);
            assertArrayEquals(test_input,
expected_output);
```

```
@Test
    public void test_case4(){
        HeapSort h_sort = new HeapSort();
        int[] test_input = {3,1,2};
        int [] expected_output = {1,2,3};
        h_sort.sort(test_input);
        assertArrayEquals(test_input,
expected_output);
    }


@Test
    public void test_case5(){
        HeapSort h_sort = new HeapSort();
        int[] test_input = {-2,-1,3,2,1,0,-2};
        int [] expected_output =
{-2,-2,-1,0,1,2,3};
        h_sort.sort(test_input);
        h_sort.printArray(expected_output);
        assertArrayEquals(test_input,
expected_output);
    }
}
```

}	

4.

a) Test_input1 = {}



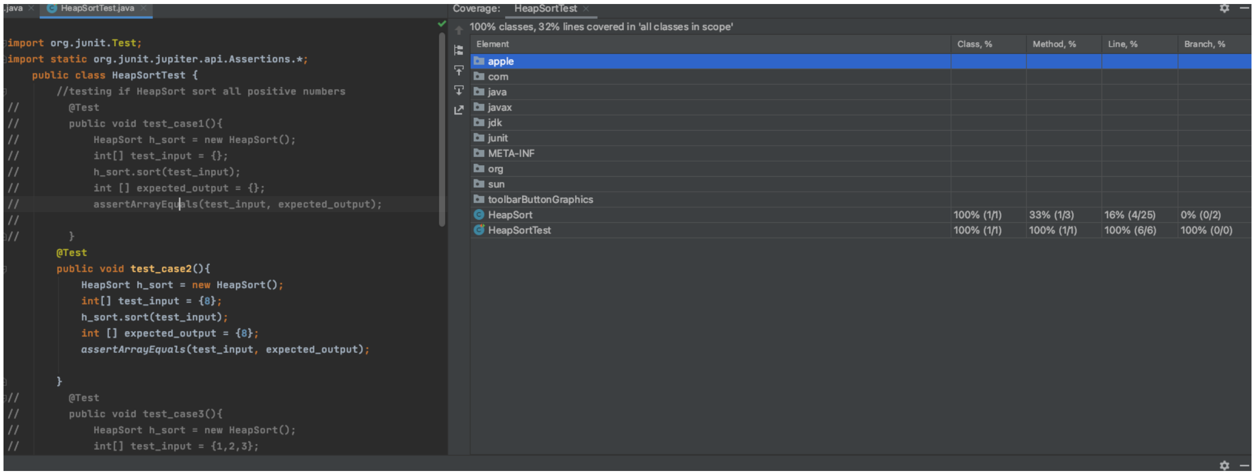
```
import org.junit.Test;
import static org.junit.jupiter.api.Assertions.*;
    public class HeapSortTest {
        //testing if HeapSort sort all positive numbers
        @Test
        public void test_case1(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {};
            h_sort.sort(test_input);
            int [] expected_output = {};
            assertArrayEquals(test_input, expected_output);

        }
//        @Test
//        public void test_case2(){
//            HeapSort h_sort = new HeapSort();
//            int[] test_input = {1,2,3};
//            int [] expected_output = {1,2,3};
//            h_sort.sort(test_input);
//            h_sort.printArray(expected_output);
//            assertArrayEquals(test_input, expected_output);
//
//        }
//        @Test
//        public void test_case3(){
//            HeapSort h_sort = new HeapSort();
```

100% classes, 32% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
apple				
com				
java				
javax				
jdk				
junit				
META-INF				
org				
sun				
toolbarButtonGraphics				
HeapSort	100% (1/1)	33% (1/3)	16% (4/25)	0% (0/2)
HeapSortTest	100% (1/1)	100% (1/1)	100% (6/6)	100% (0/0)

Testcase1: when array is empty, we see that the code exits just in the beginning, and there is very little method, line, branch coverage

b) Test_input2 = {8}



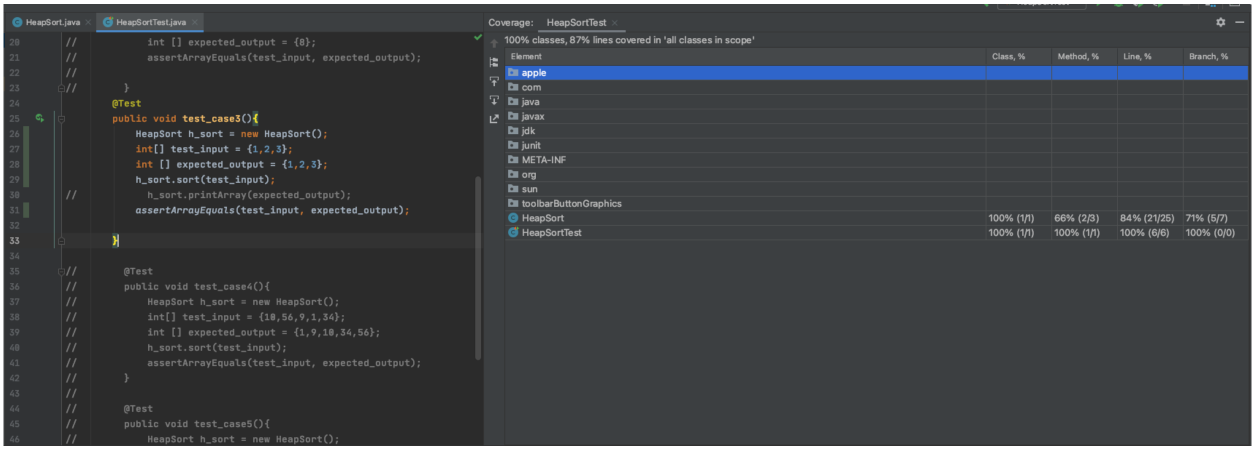
```
import org.junit.Test;
import static org.junit.jupiter.api.Assertions.*;
    public class HeapSortTest {
        //testing if HeapSort sort all positive numbers
//        @Test
//        public void test_case1(){
//            HeapSort h_sort = new HeapSort();
//            int[] test_input = {};
//            h_sort.sort(test_input);
//            int [] expected_output = {};
//            assertArrayEquals(test_input, expected_output);
//
//        }
        @Test
        public void test_case2(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {8};
            h_sort.sort(test_input);
            int [] expected_output = {8};
            assertArrayEquals(test_input, expected_output);

        }
//        @Test
//        public void test_case3(){
//            HeapSort h_sort = new HeapSort();
//            int[] test_input = {1,2,3};
```

100% classes, 32% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
apple				
com				
java				
javax				
jdk				
junit				
META-INF				
org				
sun				
toolbarButtonGraphics				
HeapSort	100% (1/1)	33% (1/3)	16% (4/25)	0% (0/2)
HeapSortTest	100% (1/1)	100% (1/1)	100% (6/6)	100% (0/0)

Testcase2: when array has single element, we still see that the code exits just in the beginning, and there is no improvement method, line, branch coverage

c) Test_input3 = {1,2,3}



```
//          int [] expected_output = {8};
//          assertArrayEquals(test_input, expected_output);
//
//      }
        @Test
        public void test_case3(){
            HeapSort h_sort = new HeapSort();
            int[] test_input = {1,2,3};
            int [] expected_output = {1,2,3};
            h_sort.sort(test_input);
//            h_sort.printArray(expected_output);
            assertArrayEquals(test_input, expected_output);

        }

//      @Test
//      public void test_case4(){
//          HeapSort h_sort = new HeapSort();
//          int[] test_input = {10,56,9,1,34};
//          int [] expected_output = {1,9,10,34,56};
//          h_sort.sort(test_input);
//          assertArrayEquals(test_input, expected_output);
//      }
//
//      @Test
//      public void test_case5(){
//          HeapSort h_sort = new HeapSort();
```

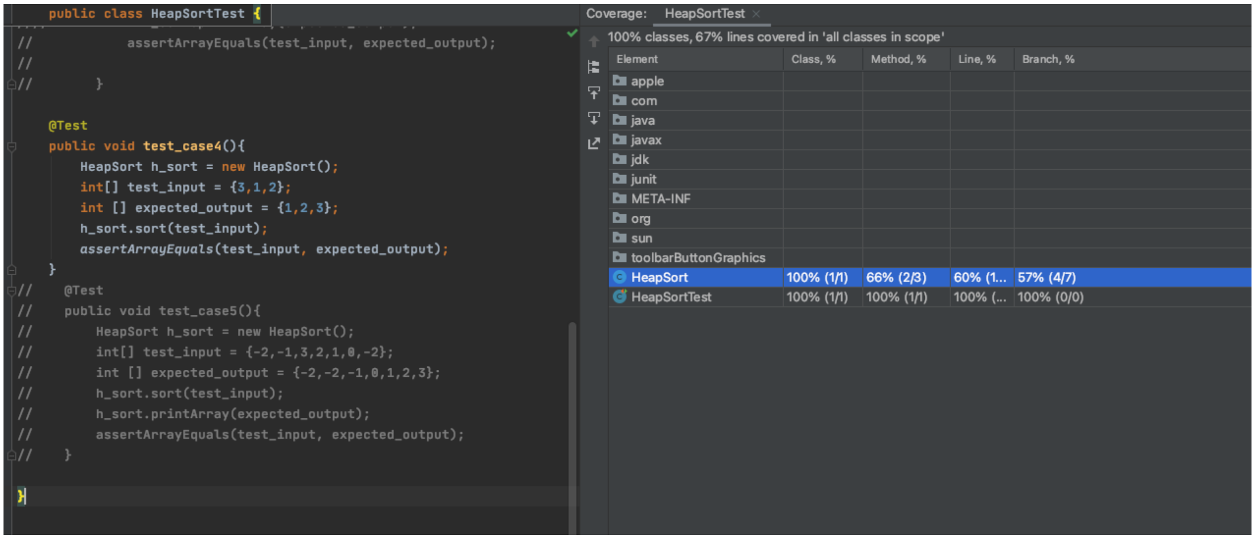
Coverage: HeapSortTest

100% classes, 87% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
apple				
com				
java				
javax				
jdk				
junit				
META-INF				
org				
sun				
toolbarButtonGraphics				
HeapSort	100% (1/1)	66% (2/3)	84% (21/25)	71% (5/7)
HeapSortTest	100% (1/1)	100% (1/1)	100% (6/6)	100% (0/0)

Testcase3: when the array is already sorted, we now see there is improvement in method, line, branch coverage

d) **Test_input4 = {3,1,2}**



```
public class HeapSortTest {
//          assertArrayEquals(test_input, expected_output);
//
//      }

    @Test
    public void test_case4(){
        HeapSort h_sort = new HeapSort();
        int[] test_input = {3,1,2};
        int [] expected_output = {1,2,3};
        h_sort.sort(test_input);
        assertArrayEquals(test_input, expected_output);
    }
//    @Test
//    public void test_case5(){
//        HeapSort h_sort = new HeapSort();
//        int[] test_input = {-2,-1,3,2,1,0,-2};
//        int [] expected_output = {-2,-2,-1,0,1,2,3};
//        h_sort.sort(test_input);
//        h_sort.printArray(expected_output);
//        assertArrayEquals(test_input, expected_output);
//    }

}
```

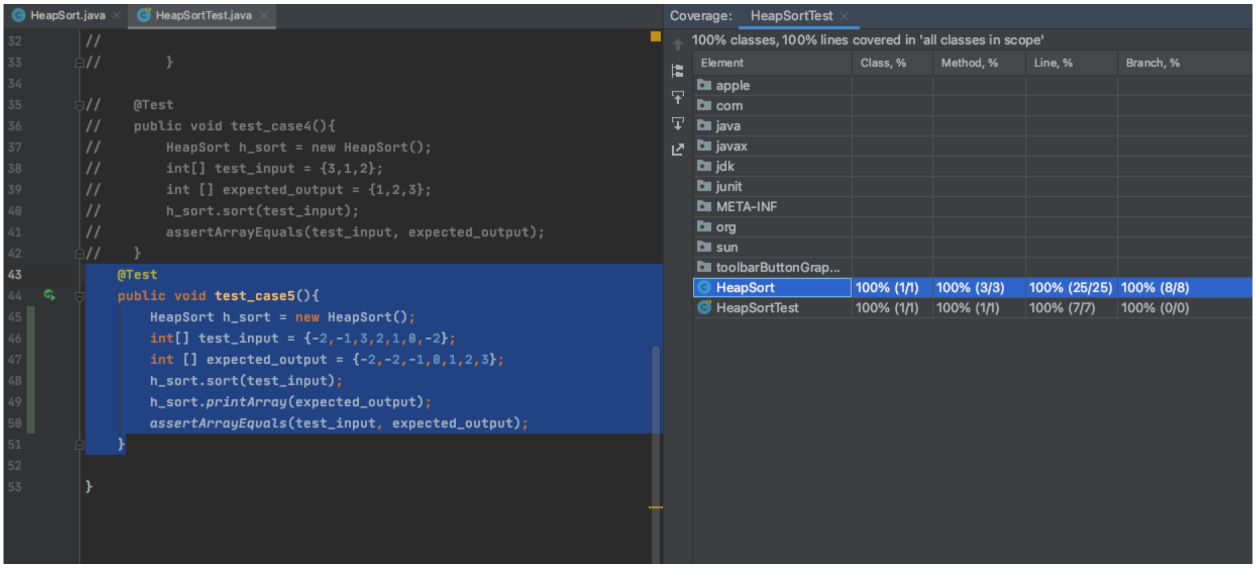
Coverage: HeapSortTest

100% classes, 67% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
apple				
com				
java				
javax				
jdk				
junit				
META-INF				
org				
sun				
toolbarButtonGraphics				
HeapSort	100% (1/1)	66% (2/3)	60% (1...	57% (4/7)
HeapSortTest	100% (1/1)	100% (1/1)	100% (...	100% (0/0)

Testcase4: when a unsorted positive integer array is taken, we see 57% branch coverage, while the line is 60% and methods are at 66%

e) **Test_input5 = {-2,-1,3,2,1,0,-2}**



```java
//
//        }

//    @Test
//    public void test_case4(){
//        HeapSort h_sort = new HeapSort();
//        int[] test_input = {3,1,2};
//        int [] expected_output = {1,2,3};
//        h_sort.sort(test_input);
//        assertArrayEquals(test_input, expected_output);
//    }
    @Test
    public void test_case5(){
        HeapSort h_sort = new HeapSort();
        int[] test_input = {-2,-1,3,2,1,0,-2};
        int [] expected_output = {-2,-2,-1,0,1,2,3};
        h_sort.sort(test_input);
        h_sort.printArray(expected_output);
        assertArrayEquals(test_input, expected_output);
    }

}
```

100% classes, 100% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
apple				
com				
java				
javax				
jdk				
junit				
META-INF				
org				
sun				
toolbarButtonGrap...				
HeapSort	100% (1/1)	100% (3/3)	100% (25/25)	100% (8/8)
HeapSortTest	100% (1/1)	100% (1/1)	100% (7/7)	100% (0/0)

Testcase5: when an unsorted positive & negative integers array is taken, and the printArray method is invoked we see 100% branch coverage, 100% line coverage and 100% method coverage.

Evaluation of the tool's usefulness:
This plugin helps us to identify the lines of code which are not executed,methods which are not invoked, branches which are not covered, finally helping us to identify the deadcode and both conditional as well as unconditional branching It not only provides the coverage in percentage, but also provides the estimate of the exact number of lines, methods, classes, branches executed out of the total number. It highlights the lines, methods covered in green and uncovered parts in red, which makes it easier to figure out the deadcode. It also gives an option to select the specific part of the file in the project for code coverage. To explore further, One can also view detailed code coverage reports on demand.

Static source code analysis tool description: I am using PMD plugin by embedding it into IntelliJ. It is a static source code analysis tool. It helps us to find the data anomalies like unused variables, catch blocks which are empty, and unnecessary creation of objects. One can select any type of predefined analysis one wants or write his own custom rules.

The PMD plugin provides the following type of analysis:

- Best practices
- Code style
- Design
- Documentation
- Error prone

- Multithreading
- Performance

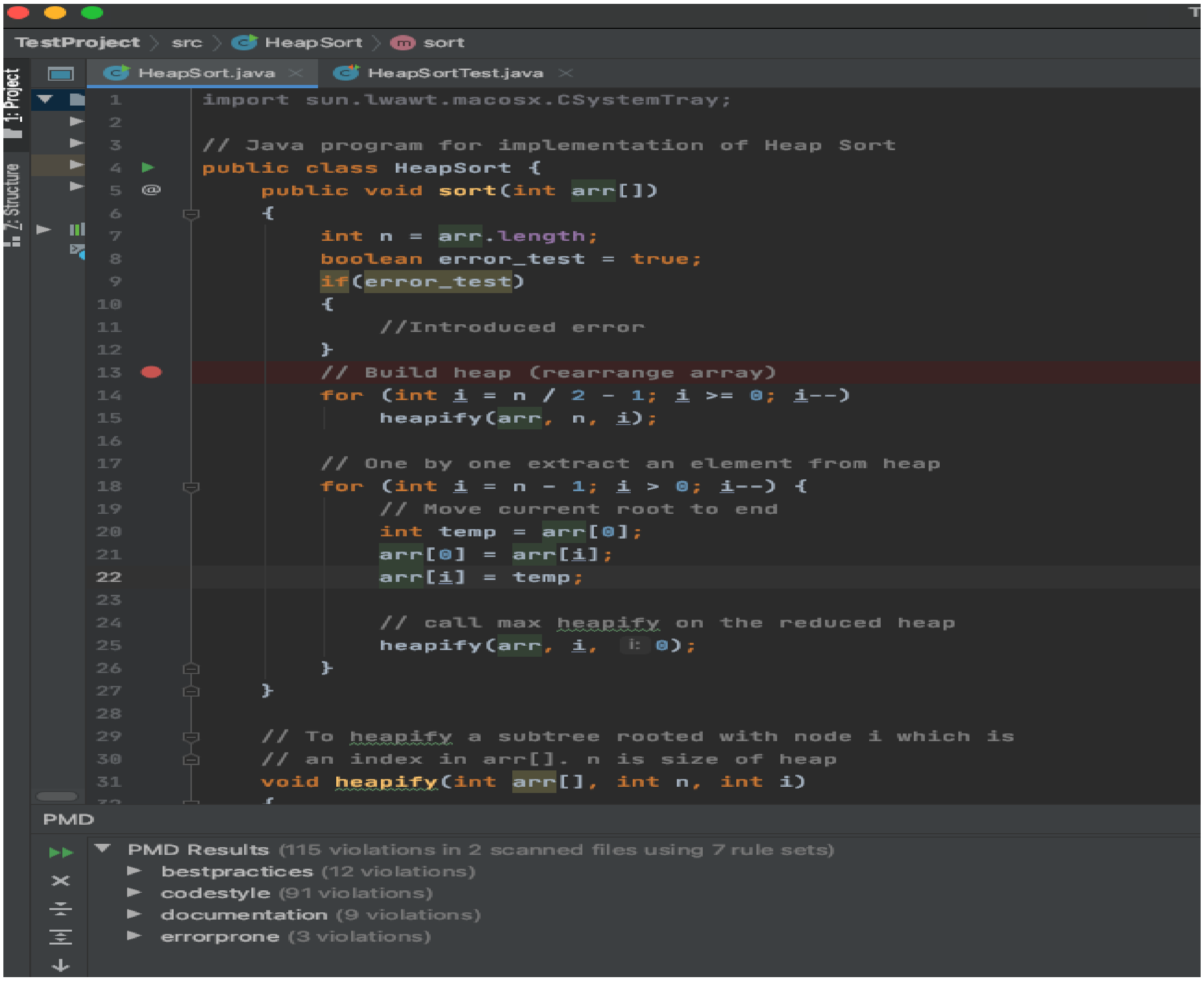
Source Listing:

```java
import sun.lwawt.macosx.CSystemTray;

// Java program for implementation of Heap Sort
public class HeapSort {
    public void sort(int arr[])
    {
        int n = arr.length;
        boolean error_test = true;
        if(error_test)
        {
            //Introduced error
        }
        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }


    void heapify(int arr[], int n, int i)
    {
```

```java
       int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }



// Driver code
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int n = arr.length;

        HeapSort ob = new HeapSort();
```

	ob.sort(arr);
	
	System.out.println("Sorted array is");
	printArray(arr);
	}
	}

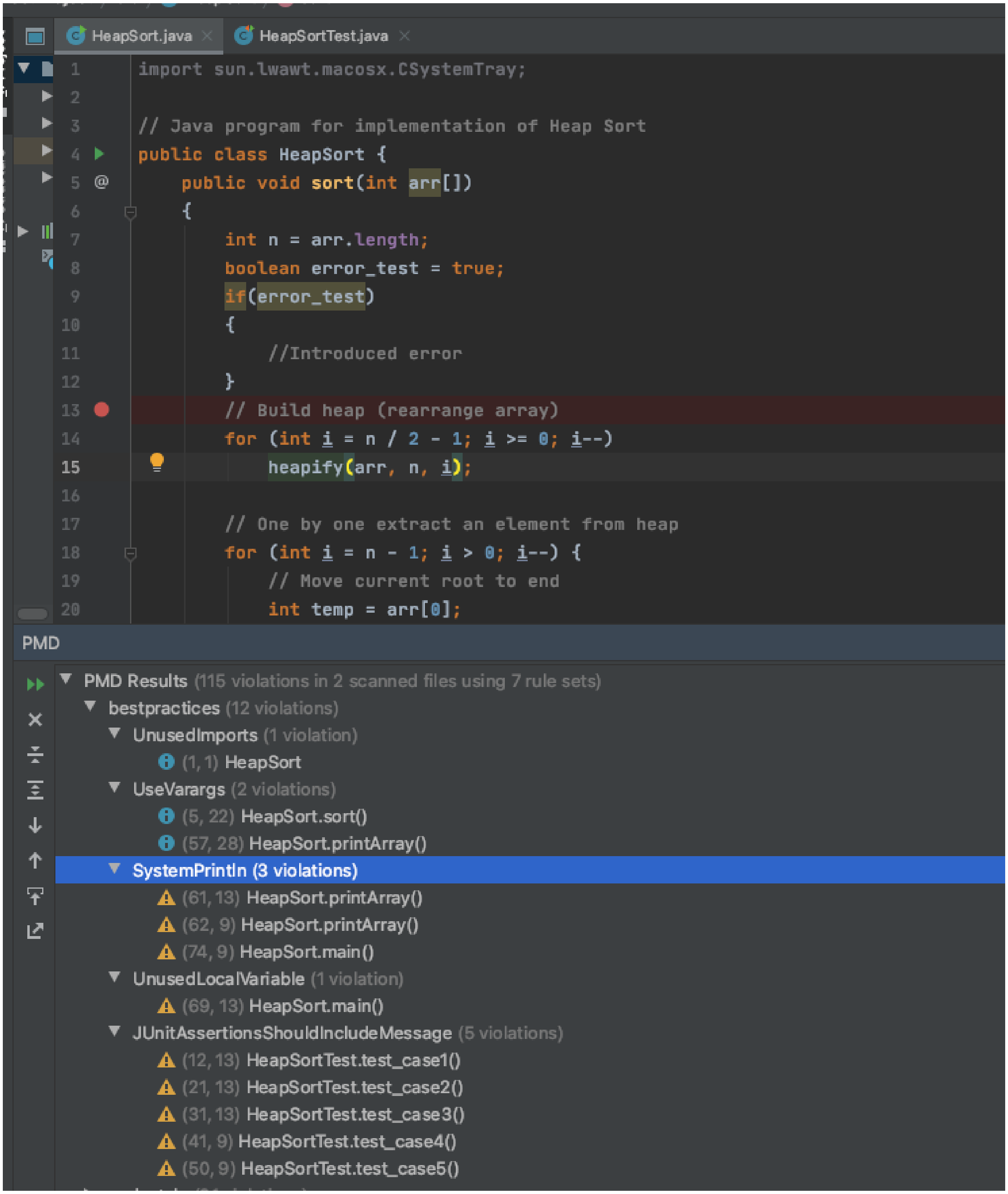
Screenshots of types of analysis:



```
import sun.lwawt.macosx.CSystemTray;

// Java program for implementation of Heap Sort
public class HeapSort {
    public void sort(int arr[])
    {
        int n = arr.length;
        boolean error_test = true;
        if(error_test)
        {
            //Introduced error
        }
        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i,  i: 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
```

```
PMD
▼ PMD Results (115 violations in 2 scanned files using 7 rule sets)
    ► bestpractices (12 violations)
    ► codestyle (91 violations)
    ► documentation (9 violations)
    ► errorprone (3 violations)
```

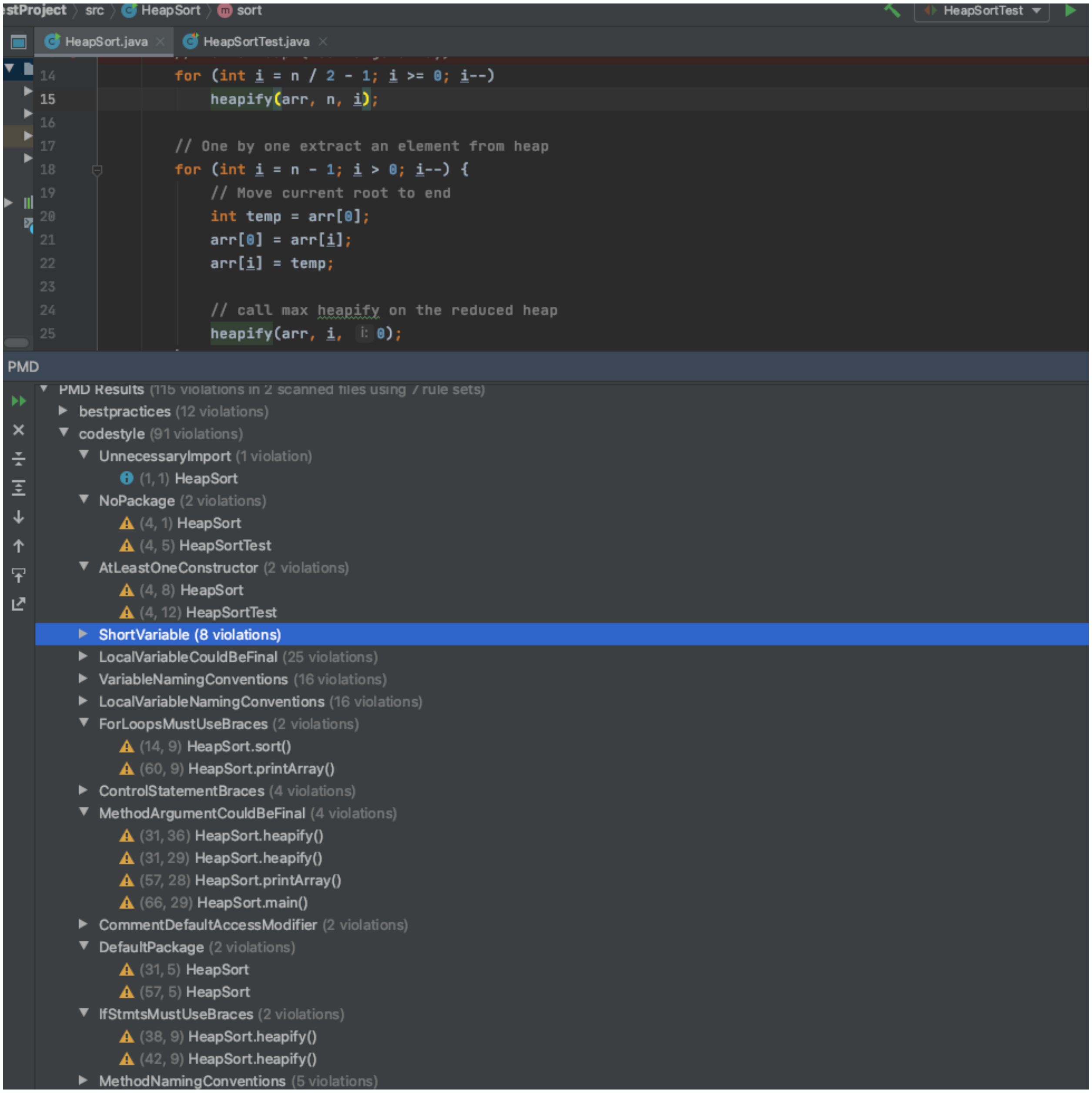
Best Practices:



```java
import sun.lwawt.macosx.CSystemTray;

// Java program for implementation of Heap Sort
public class HeapSort {
    public void sort(int arr[])
    {
        int n = arr.length;
        boolean error_test = true;
        if(error_test)
        {
            //Introduced error
        }
        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
```

```
▼ PMD Results (115 violations in 2 scanned files using 7 rule sets)
  ▼ bestpractices (12 violations)
    ▼ UnusedImports (1 violation)
        (1, 1) HeapSort
    ▼ UseVarargs (2 violations)
        (5, 22) HeapSort.sort()
        (57, 28) HeapSort.printArray()
    ▼ SystemPrintln (3 violations)
        (61, 13) HeapSort.printArray()
        (62, 9) HeapSort.printArray()
        (74, 9) HeapSort.main()
    ▼ UnusedLocalVariable (1 violation)
        (69, 13) HeapSort.main()
    ▼ JUnitAssertionsShouldIncludeMessage (5 violations)
        (12, 13) HeapSortTest.test_case1()
        (21, 13) HeapSortTest.test_case2()
        (31, 13) HeapSortTest.test_case3()
        (41, 9) HeapSortTest.test_case4()
        (50, 9) HeapSortTest.test_case5()
```

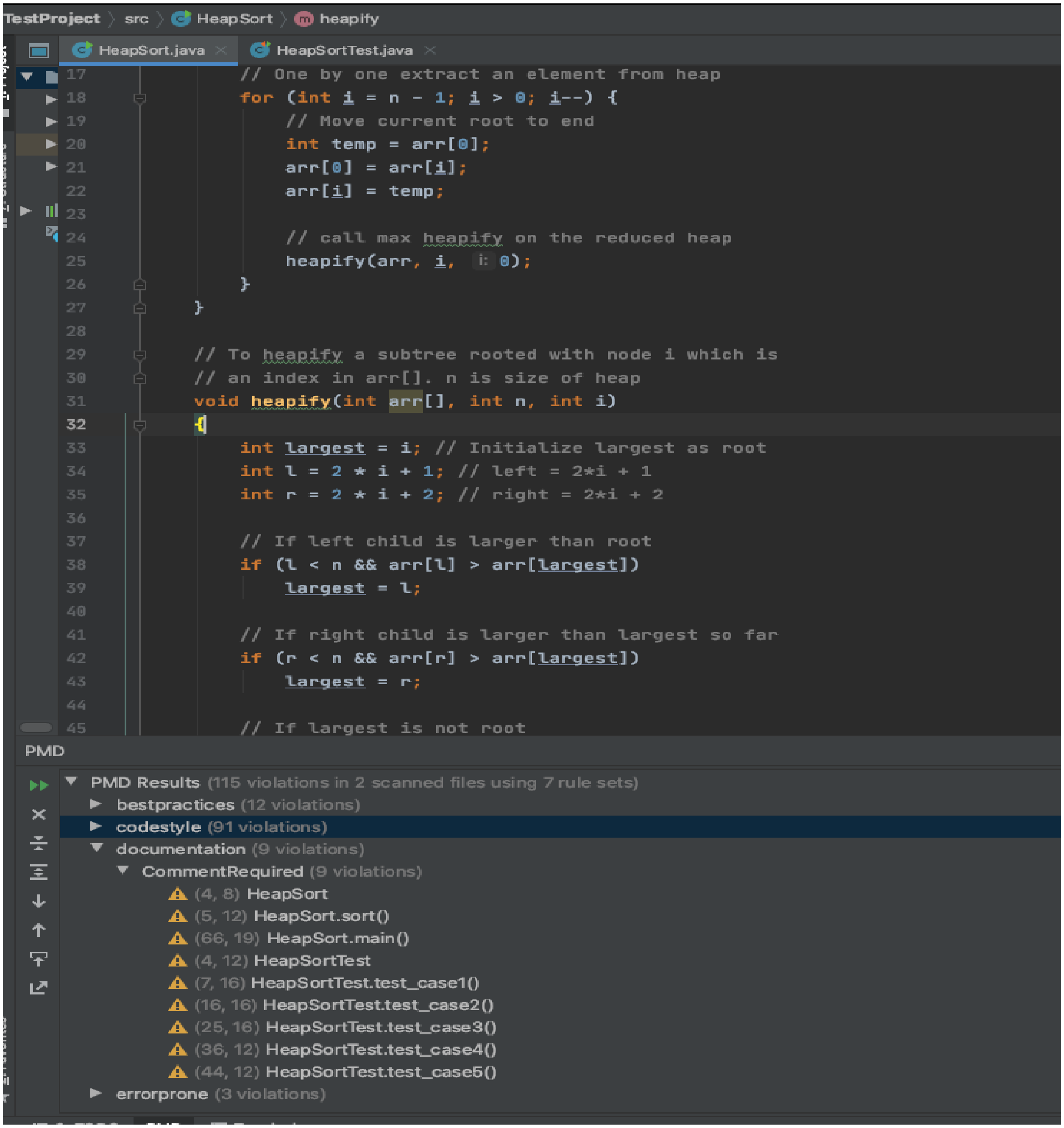
Code Style:



```java
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i,  i: 0);
```

```
▼ PMD Results (115 violations in 2 scanned files using 7 rule sets)
  ▶ bestpractices (12 violations)
  ▼ codestyle (91 violations)
    ▼ UnnecessaryImport (1 violation)
        (1, 1) HeapSort
    ▼ NoPackage (2 violations)
        (4, 1) HeapSort
        (4, 5) HeapSortTest
    ▼ AtLeastOneConstructor (2 violations)
        (4, 8) HeapSort
        (4, 12) HeapSortTest
    ▶ ShortVariable (8 violations)
    ▶ LocalVariableCouldBeFinal (25 violations)
    ▶ VariableNamingConventions (16 violations)
    ▶ LocalVariableNamingConventions (16 violations)
    ▼ ForLoopsMustUseBraces (2 violations)
        (14, 9) HeapSort.sort()
        (60, 9) HeapSort.printArray()
    ▶ ControlStatementBraces (4 violations)
    ▼ MethodArgumentCouldBeFinal (4 violations)
        (31, 36) HeapSort.heapify()
        (31, 29) HeapSort.heapify()
        (57, 28) HeapSort.printArray()
        (66, 29) HeapSort.main()
    ▶ CommentDefaultAccessModifier (2 violations)
    ▼ DefaultPackage (2 violations)
        (31, 5) HeapSort
        (57, 5) HeapSort
    ▼ IfStmtsMustUseBraces (2 violations)
        (38, 9) HeapSort.heapify()
        (42, 9) HeapSort.heapify()
    ▶ MethodNamingConventions (5 violations)
```

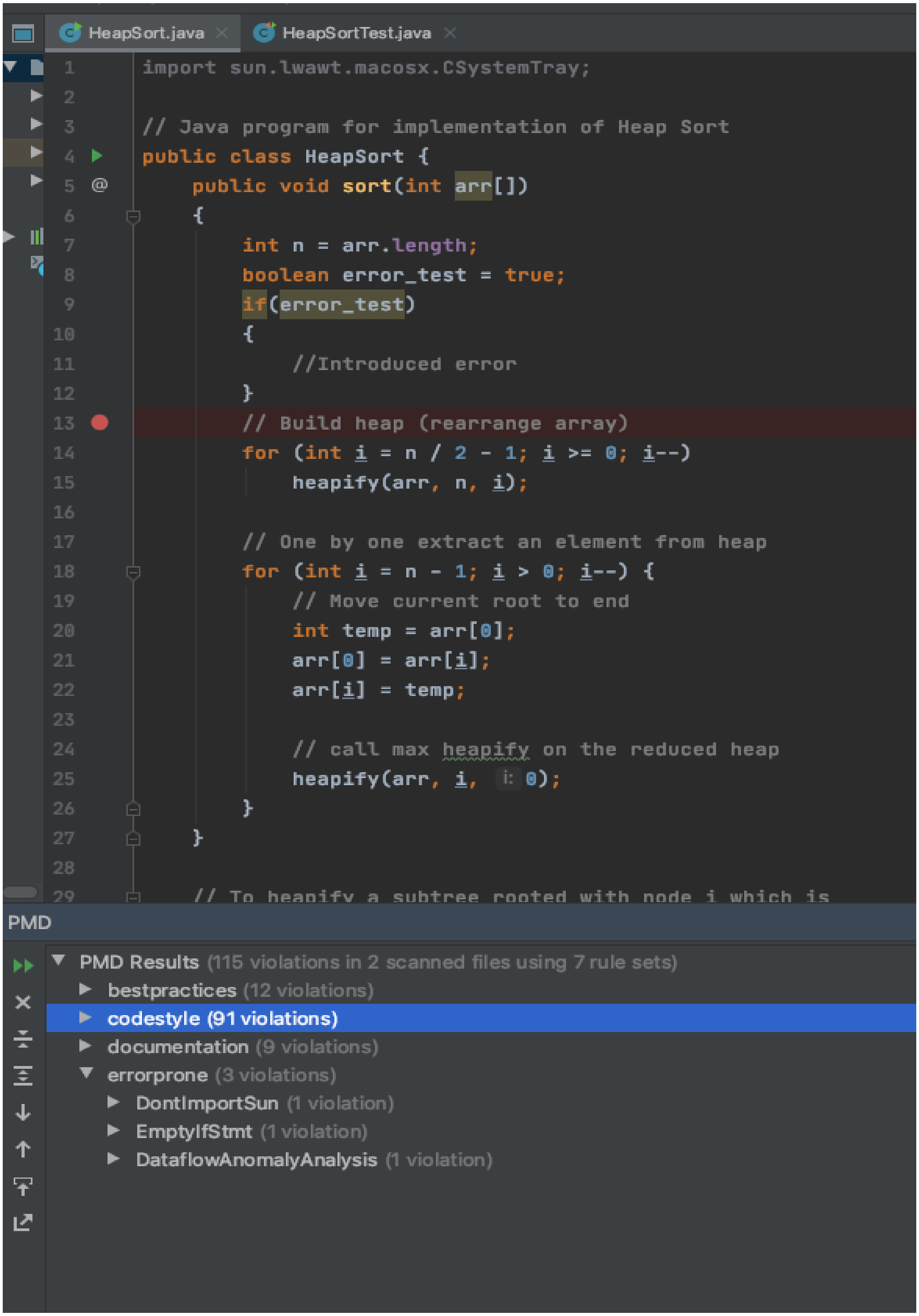
Documentation:



```java
        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i,  i: 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
```

```
▼ PMD Results (115 violations in 2 scanned files using 7 rule sets)
    ▶ bestpractices (12 violations)
    ▶ codestyle (91 violations)
    ▼ documentation (9 violations)
        ▼ CommentRequired (9 violations)
            ⚠ (4, 8) HeapSort
            ⚠ (5, 12) HeapSort.sort()
            ⚠ (66, 19) HeapSort.main()
            ⚠ (4, 12) HeapSortTest
            ⚠ (7, 16) HeapSortTest.test_case1()
            ⚠ (16, 16) HeapSortTest.test_case2()
            ⚠ (25, 16) HeapSortTest.test_case3()
            ⚠ (36, 12) HeapSortTest.test_case4()
            ⚠ (44, 12) HeapSortTest.test_case5()
    ▶ errorprone (3 violations)
```

Error Prone:



```java
import sun.lwawt.macosx.CSystemTray;

// Java program for implementation of Heap Sort
public class HeapSort {
    public void sort(int arr[])
    {
        int n = arr.length;
        boolean error_test = true;
        if(error_test)
        {
            //Introduced error
        }
        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i,  i: 0);
        }
    }

    // To heapify a subtree rooted with node i which is
```

PMD

```
▼ PMD Results (115 violations in 2 scanned files using 7 rule sets)
    ▶ bestpractices (12 violations)
    ▶ codestyle (91 violations)
    ▶ documentation (9 violations)
    ▼ errorprone (3 violations)
        ▶ DontImportSun (1 violation)
        ▶ EmptyIfStmt (1 violation)
        ▶ DataflowAnomalyAnalysis (1 violation)
```

Evaluation of the static analysis tool's usefulness:

I find the tool very helpful as it covers all the aspects of the development and design process efficiently. It provides immediate feedback. It also comes with custom rules along with the predefined rules. It can definitely improve the coding, design, and development skills. One can also easily figure out the coding errors using error-prone feedback provided by the tool. Nowadays multithreading and performance has been a vital part of many applications. This tool also provides an evaluation for multithreading and performance.

References:

https://www.geeksforgeeks.org/java-program-for-heap-sort/
https://www.jetbrains.com/help/idea/code-coverage.html
https://pmd.github.io/pmd-6.39.0/