2017

Logbook: U1364096

ALGORITHMS PROCESSES AND DATA

LUKE MCCANN

TUTOR: DR HUGH OSBOURNE | Huddersfield University

Table of Contents

Week1/2

1) SimpleRandomListing is not an efficient implementation of the abstract RandomListing class.   Design and implement a better solution. Call this class CleverRandomListing.

2) Add a timing method to the RandomListing class and use this to compare the efficiency of the two extensions of this class.

CleverRandomListing.java

```
package intArrays;

import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

import Search.IndexingError;


public class CleverRandomListing extends RandomListing{

    /*
     * Generates an array of elements of the super class.
     * @param size - the size of array to be generated.
     */
        public CleverRandomListing(int size) {
                super(size);
        }


        /**
         * (non-Javadoc)
         * @see intArrays.RandomListing#randomise()
         * randomise() - Traverses array "n",
         * gets a random index, creates a temporary variable
         * in which current position is stored.
         * switches the values of the random index and
         * the value of current position (i) in copy.
         */

        @Override
        protected void randomise() {
                int[] copy = new int[getArray().length];

                for( int i = 0; i < getArray().length; i++) {
                        int randomIndex = getRandomIndex();
                        int temp = getArray()[i];
                        copy[i] = getArray()[randomIndex];
                        copy[randomIndex] = temp;

                        getArray()[i] = copy[i];
                        }
                }



    public static void main(String[] args) {
        RandomListing count = new CleverRandomListing(1000);
        System.out.println(Arrays.toString(count.getArray()));
    }


        @Override
        public int findKthElement(int[] array, int k) throws IndexingError {
                // TODO Auto-generated method stub
                return 0;
        }


        @Override
        public double time(int[] array, int k) throws IndexingError {
                // TODO Auto-generated method stub
                return 0;
        }
} // End of class CleverRandomListing
```

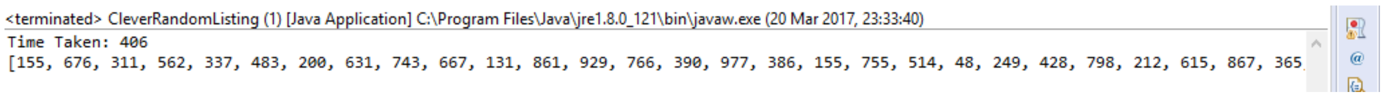
RandomListing.java

```
package intArrays;

import java.util.Arrays;
import java.util.Random;
import java.util.Timer;
import Search.IndexingError;
import Search.SearchTimer;
import Search.SimpleSearchTimer;
import Search.TimedSearch;

/**
 * An array generator that generates an array containing the values 0..n in
 * random order.  This abstract class does not specify how the contents of the
 * array are randomised - this must be implemented in concrete extensions of
 * this class.
 *
 * @author Hugh Osborne
 * @Update Luke McCann
 * @version September 2016
 */


public abstract class RandomListing extends SortedListing implements TimedSearch
{
    /**
     * Generate an array containing elements in a random order
     *
     * @param size the size of the array to be generated
     */
    public RandomListing(int size) {
        super(size);
        try {
                        timing();
                } catch (IndexingError e) {
                        e.printStackTrace();
                }
    }

    private Random random = new Random();

    /**
     * Randomise the order of the elements in the array
     */
    protected abstract void randomise();

    /**
     * A utility to provide a random index within the range of indices of the
     * array generated by this array generator.
     * @return an integer index between 0 and <tt>size</tt>-1 inclusive, where
     * <tt>size</tt> is the size of the array generated by this array generator.
     */
    protected int getRandomIndex() {
        return random.nextInt(getArray().length);
    }

    /**
     *
     * @throws IndexingError
     * Stores the current time and sends a call to randomise.
     * once randomise is executed, gets the current time and
     * minuses the startTime to get the difference between the two.
     */
```

```java
    public void timing() throws IndexingError {
        long startTime = System.nanoTime();
        randomise();
        long difference = System.nanoTime() - startTime;
        System.out.println("Time Taken: " + difference / 1000);
    }
} // End of class RandomListing
```

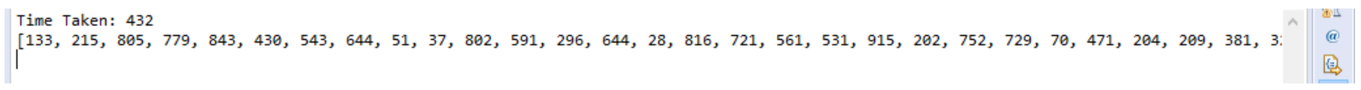
Tests

CleverRandomListing:



```
<terminated> CleverRandomListing (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (20 Mar 2017, 23:33:40)
Time Taken: 406
[155, 676, 311, 562, 337, 483, 200, 631, 743, 667, 131, 861, 929, 766, 390, 977, 386, 155, 755, 514, 48, 249, 428, 798, 212, 615, 867, 365
```

RandomListing:



```
Time Taken: 432
[133, 215, 805, 779, 843, 430, 543, 644, 51, 37, 802, 591, 296, 644, 28, 816, 721, 561, 531, 915, 202, 752, 729, 70, 471, 204, 209, 381, 3
```

Timed Test Results

Class	Time Taken (milliseconds)
CleverRandomListing	796
SimpleRandomListing	2277

CleverRandomListing has a shorter execution time when compared to RandomListing. On average CleverRandomListing outperforms RandomListing in the majority of situations when it is run, along with this it also significantly outperforms SimpleRandomListing as seen from the above comparison.

Additional Questions

1) The SimpleRandomListing class contains syntactic and semantic errors, correct these errors, add sufficient comments.

2) Now use SimpleRandomListing and the time methods of the SimpleSearchTimer and CleverSearchTimer to compare the efficiency of the two implementations of the TimedSearch class

SimpleRandomListing.java

```java
package intArrays;

import java.util.Arrays; // in order to be able to use the fill(...) method
/**
 * An extension of RandomCount
 *
 * @author Hugh Osborne
 * @edited Luke McCann U1364096
 * @version September 2016
 */
public class SimpleRandomListing extends RandomListing
{
    /**
     * Generates an array of elements of the super class.
     * @param size - the size of array to be generated.
     */
    public SimpleRandomListing(int size) {
        super(size);
    }

    /**
     * Creates a copy of the Array.
     * Randomises the order of the copy and saves the copy to the Array.
     * @return - and integer between 0 and size, where size is the
     * length of the array itself.
     */
    protected void randomise() {
        int[] copy = new int[getArray().length];
        // used to indicate if elements have been used
        boolean[] used = new boolean[getArray().length];
        Arrays.fill(used,false);
        for (int index = 0; index < getArray().length; index++) {
                int randomIndex;
                do {
                 randomIndex = getRandomIndex();
            } while (used[randomIndex]);
            copy[index] = getArray()[randomIndex];
            used[randomIndex] = true;
        }
        for (int index = 0; index < getArray().length; index++) {
            getArray()[index] = copy[index];
        }
    }

    public static void main(String[] args) {
        RandomListing count = new SimpleRandomListing(100);
        System.out.println(Arrays.toString(count.getArray()));
    }

    @Override
    public int findKthElement(int[] array, int k) throws IndexingError {
         // TODO Auto-generated method stub
         return 0;
        }

    @Override
    public double time(int[] array, int k) throws IndexingError {
         // TODO Auto-generated method stub
         return 0;
        }

} // End of class SimpleRandomListing
```

SimpleSearchTimer.java

```
package Search;

import static org.junit.Assert.*;
import java.util.Arrays;
import java.util.Collection;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;
import org.junit.rules.TestName;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameter;
import org.junit.runners.Parameterized.Parameters;
import intArrays.SimpleRandomListing;

/*
```

```
 * SimpleSearchTimerTest - Test Class for SimpleSearchTimer
 * Author - Luke McCann U1364096
 * Created: October 2016
 */

public class SimpleSearchTimerTest {

        private SimpleRandomListing testArray = new SimpleRandomListing(25);
        private int[] fourArray = { 1, 3, 6, 4 };
        private SimpleSearchTimer findKTest = new SimpleSearchTimer();

        /*
         * @Rule - Creates a new testName to allow it to be printed.
         */
        @Rule public TestName testName = new TestName();

        /*
         * findKthLargest
         * @param int k - which largest number you want to find.
         * @param int n - the size of array to be generated.
         */
        public void findKthLargest(int k, int n) throws IndexingError {
            int output = findKTest.findKthElement(new SimpleRandomListing(n).getArray(), k);
            equals(n - k);
            System.out.println("Test \"" + testName.getMethodName() + ": " + output );
        }

        @Test
        public void testArrayTen() throws IndexingError {
                findKthLargest(2, 10);
        }

        @Test
        public void testArrayTwenty() throws IndexingError {
                findKthLargest(5, 20);
        }

        @Test
        public void testArrayThirtySix() throws IndexingError {
                findKthLargest(9, 36);
        }

        /*
         * testFindFourthElementOfTwentyFive -
         * Finds the fourth largest element from an array of twenty five elements.
         */
        @Test
        public void testFindFourthElementOfTwentyFive() throws IndexingError {
                int output = findKTest.findKthElement(testArray.getArray(), 4);
                assertEquals(21, output);
                System.out.println("Test \"" + testName.getMethodName() + ": " + output);
        }

        /*
         * testFindSixthElementOfFour -
         * Attempts to find the Sixth element in an array of only four.
         * @throws Indexing Error
         */
        @Rule
        public final ExpectedException indexError= ExpectedException.none();

        @Test
        public void testFindSixthElementOfFour() throws IndexingError {
                try {
                    indexError.expect(Search.IndexingError.class);
                    findKTest.findKthElement(fourArray, 6);
                }
                catch(IndexingError e) {
                        System.err.println("Test \"" + testName.getMethodName() + ": " +
"Indexing Error!");

                }
        }
} // End of class SimpleSearchTimerTest
```

Week3/4

1) Write a generic method to exchange two elements of an array. The method should take an array and two integer indices into the array, and swap the two entries in the array at those indices.

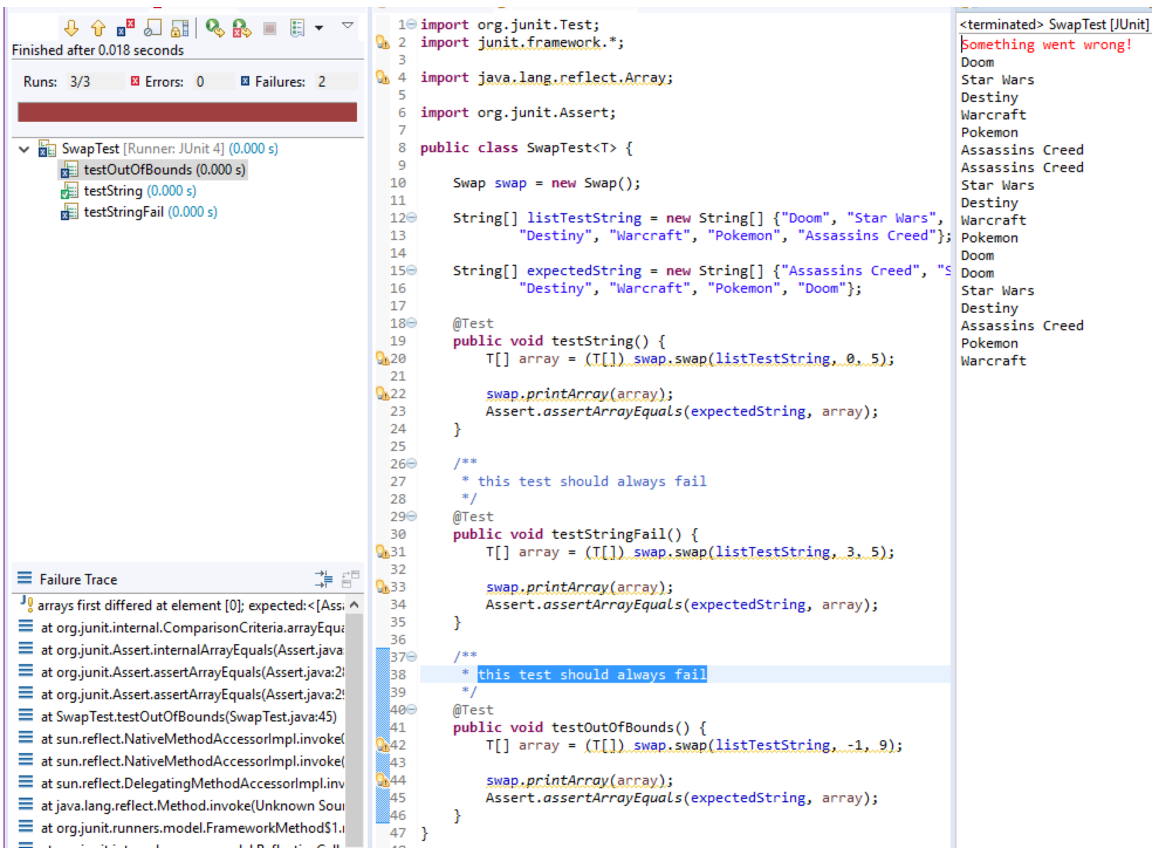
Swap.java

```java
import java.lang.reflect.Array;

/**
 *
 * @author Luke McCann
 * Date October 2016
 *
 * @Swap swaps two specified element positions
 * in a generic array.
 */

public class Swap {

	public static void main(String[] args) {
		String[] listTestString = new String[] {"Doom", "Star Wars", "Destiny",
"Warcraft", "Pokemon", "Assassins Creed"};

		swap(listTestString, 2, 5);
	}

	/*
	 * Swaps the position of two specified elements.
	 * @param - T[] array of elements.
	 * int a - element a's position.
	 * int b - element to swap withs position.
	 */
	public static <T> void swap(T[] array, int a, int b ) {
		try {
			T temp = array[a];
			array[a] = array[b];
			array[b] = temp;

			printArray(array);
		}
		catch(Exception e) {
			System.err.println("Something went wrong!");
		}
	}

	/*
	 * prints the items stored in a generic array.
	 * @param array - T[] array to be printed.
	 */
	public static <T> void printArray(T[] array) {
		try {
			for(int i = 0; i < array.length; i++) {
				System.out.println(array[i]);
			}
		}
		catch(IndexOutOfBoundsException e) {
			System.err.println("Index Out Of Bound.");
		}
	}
} // End of class Swap
```

SwapTest.java

```java
import org.junit.Test;
import junit.framework.*;

import java.lang.reflect.Array;
import org.junit.Assert;

public class SwapTest<T> {

        Swap swap = new Swap();

        String[] listTestString = new String[] {"Doom", "Star Wars",
                        "Destiny", "Warcraft", "Pokemon", "Assassins Creed"};

        String[] expectedString = new String[] {"Assassins Creed", "Star Wars",
                        "Destiny", "Warcraft", "Pokemon", "Doom"};

        @Test
        public void testString() {
                T[] array = (T[]) swap.swap(listTestString, 0, 5);

                swap.printArray(array);
                Assert.assertArrayEquals(expectedString, array);
        }

        /**
         * this test should always fail
         */
        @Test
        public void testStringFail() {
                T[] array = (T[]) swap.swap(listTestString, 3, 5);

                swap.printArray(array);
                Assert.assertArrayEquals(expectedString, array);
        }

        /**
         * this test should always fail
         */
        @Test
        public void testOutOfBounds() {
                T[] array = (T[]) swap.swap(listTestString, -1, 9);

                swap.printArray(array);
                Assert.assertArrayEquals(expectedString, array);
        }
}
```

Output



Finished after 0.018 seconds

Runs: 3/3 Errors: 0 Failures: 2

SwapTest [Runner: JUnit 4] (0.000 s)
testOutOfBounds (0.000 s)
testString (0.000 s)
testStringFail (0.000 s)

Failure Trace

```
arrays first differed at element [0]; expected:<[Ass
at org.junit.internal.ComparisonCriteria.arrayEqua
at org.junit.Assert.internalArrayEquals(Assert.java
at org.junit.Assert.assertArrayEquals(Assert.java:2
at org.junit.Assert.assertArrayEquals(Assert.java:2
at SwapTest.testOutOfBounds(SwapTest.java:45)
at sun.reflect.NativeMethodAccessorImpl.invoke(
at sun.reflect.NativeMethodAccessorImpl.invoke(
at sun.reflect.DelegatingMethodAccessorImpl.inv
at java.lang.reflect.Method.invoke(Unknown Sou
at org.junit.runners.model.FrameworkMethod$1.
```

```java
import org.junit.Test;
import junit.framework.*;

import java.lang.reflect.Array;

import org.junit.Assert;

public class SwapTest<T> {

    Swap swap = new Swap();

    String[] listTestString = new String[] {"Doom", "Star Wars",
            "Destiny", "Warcraft", "Pokemon", "Assassins Creed"};

    String[] expectedString = new String[] {"Assassins Creed", "S
            "Destiny", "Warcraft", "Pokemon", "Doom"};

    @Test
    public void testString() {
        T[] array = (T[]) swap.swap(listTestString, 0, 5);

        swap.printArray(array);
        Assert.assertArrayEquals(expectedString, array);
    }

    /**
     * this test should always fail
     */
    @Test
    public void testStringFail() {
        T[] array = (T[]) swap.swap(listTestString, 3, 5);

        swap.printArray(array);
        Assert.assertArrayEquals(expectedString, array);
    }

    /**
     * this test should always fail
     */
    @Test
    public void testOutOfBounds() {
        T[] array = (T[]) swap.swap(listTestString, -1, 9);

        swap.printArray(array);
        Assert.assertArrayEquals(expectedString, array);
    }
}
```

<terminated> SwapTest [JUnit]

```
Something went wrong!
Doom
Star Wars
Destiny
Warcraft
Pokemon
Assassins Creed
Assassins Creed
Star Wars
Destiny
Warcraft
Pokemon
Doom
Doom
Star Wars
Destiny
Assassins Creed
Pokemon
Warcraft
```

Week5

1) Use implementations to time the execution of (at least) two sorting algorithms for various sizes of array and plot the results on a graph. Arrive at (approximate) formulæ for how the execution times vary in relation to the data size?

The Quicksort has a Big O formula of O (n log n), this is because we have to do n -1 comparisons for each generation and log n as we have to divide the list log n times, in other words the time taken to execute is proportionate to the input.

SelectionSort has a formula however of O(n^2) this means when we run the algorithm, we expect it will take n to the power of 2 times, this means that the algorithm requires much more additional memory, these algorithms also tend to be much slower than other more efficient algorithms.

Quicksort.java

```java
package arraySorter;

import java.util.ArrayList;

/**
 *
 * @author Luke McCann - U1364096
 * sorts an array using the QuickSort algorithm.
 * @param <T>
 */
public class QuickSort<T extends Comparable <T>> extends ArraySortTool<T> {

        NullCheck check = new NullCheck();

        @Override
        public void sort(T[] array) {
                if(check.checkNotNull(array));
                quicksort(array, 0, array.length - 1);
        }

        /*
         * sorts an array using the Quicksort algorithm
         * selects a pivot and sets the left and right pointers,
         * if the current element is larger it is left to the right
         * otherwise it is moved to the left the pivot is placed between
          * the two sides (larger and smaller than) it is then in its sorted position
         */
        private void quicksort(T[] array, int leftPointer, int rightPointer) {

                if(leftPointer >= rightPointer) {
                        return;
                }

                T pivot = array[leftPointer + (rightPointer - leftPointer)/2];

                int i = leftPointer;
                int j = rightPointer;

                while (i <= j) {

                        while(array[i].compareTo(pivot) < 0) {
                                i++;
                        }

                        while(array[j].compareTo(pivot) > 0) {
                                j--;
                        }

                        if(i <= j) {
                                T temp = array[i];
                                array[i] = array[j];
```

```java
                              array[j] = temp;

                              i++;
                              j--;
                       }
               }
               quicksort(array, leftPointer, j);
               quicksort(array, i, rightPointer);
       }
}
```

SelectionSort.java

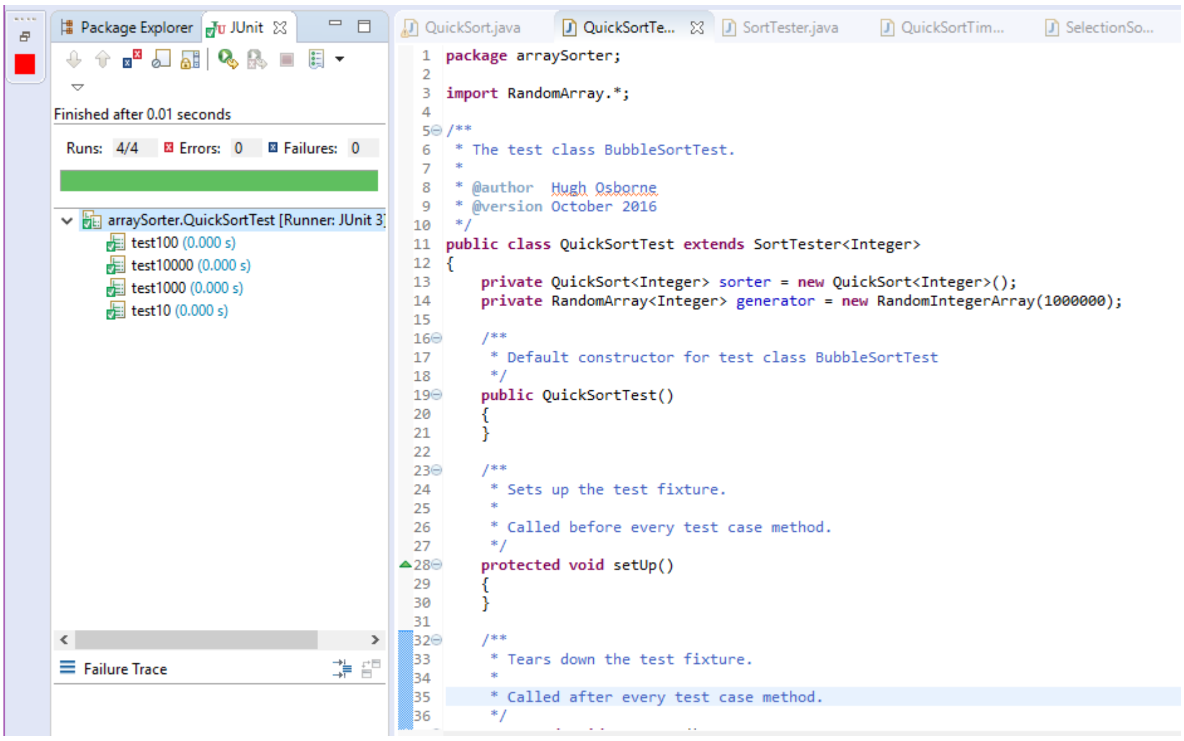
```java
package arraySorter;

/**
 *
 * @author Luke McCann - U1364096
 * Class SelectionSort - Sorts an array using the Selection sort algorithm,
 * the list is divided into two portions (sorted and unsorted) at each step
 * a item is moved from the unsorted to sorted portion, until the array is sorted.
 * @param <T>
 */
public class SelectionSort<T extends Comparable <T>> extends ArraySortTool<T> {

        NullCheck check = new NullCheck();

        /*
         * sort
         * @param T[] array - generic object of the array to be sorted
         * Checks the array is populated and sorts it using
         * the SelectionSort Algorithm.
         */
        @Override
        public void sort(T[] array) {
                if(check.checkNotNull(array));
            for (int i = 0; i < array.length - 1; i++)  {
                int minIndex = i;
                for (int j = i + 1; j < array.length; j++) {
                    if (array[j].compareTo(array[minIndex]) < 0) {
                         minIndex = j;
                         }
                    }
                T temp = array[i];
                array[i] = array[minIndex];
                array[minIndex] = temp;
               //System.out.println(array[i]);
            }
        }
}
```

NullCheck.java

```java
package arraySorter;

/**
 *
 * @author Luke McCann - U1364096
 * Class NullCheck - checks that an array is not null
 *
 */
public class NullCheck {

        /*
         * checks that an array is not null
         * or asks the user to ensure the array contains at least 2 elements
         * @param array - the array to be checked
         */
        @SuppressWarnings("unused")
        public <T> boolean checkNotNull(T[] array) {
                if(array.length <= 1 || array == null) {
                        System.err.println("Please ensure the array contains at least 2 
elements.");
                        return false;
                }
                return true;
        }
}
```

Tests

QuickSort

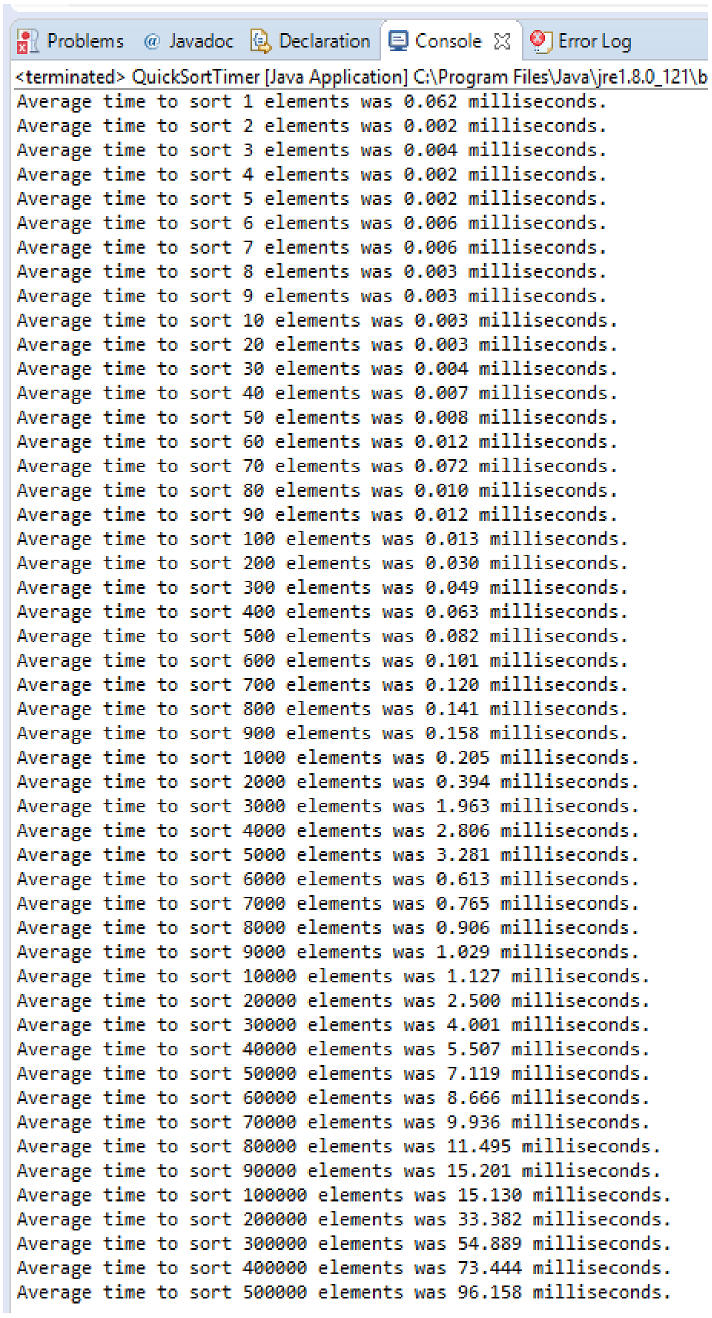


Finished after 0.01 seconds

Runs: 4/4   Errors: 0   Failures: 0

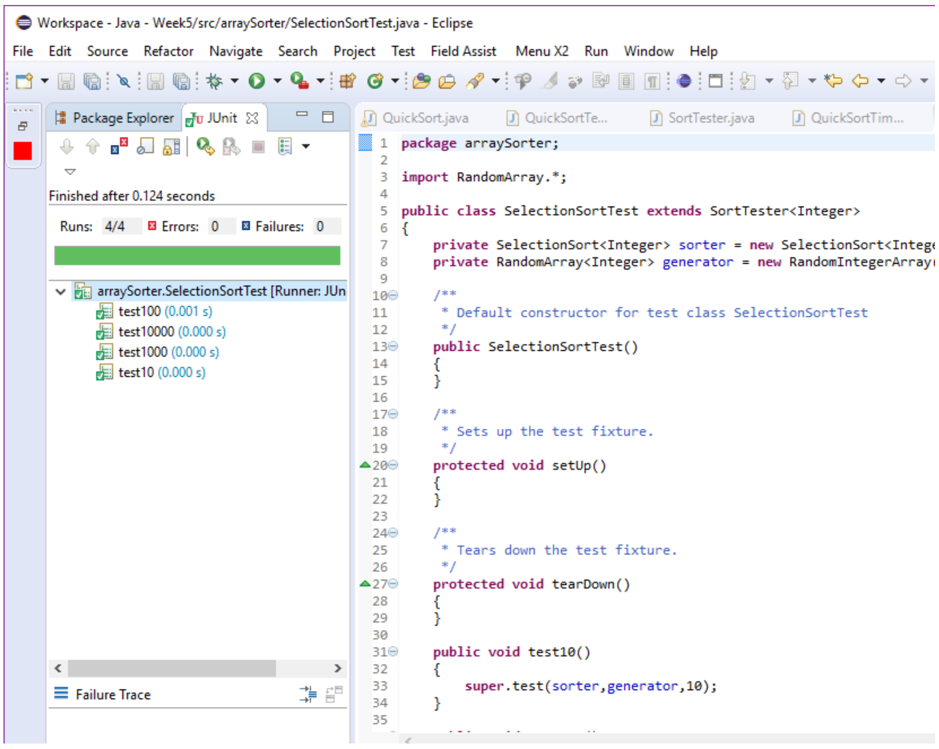
```
arraySorter.QuickSortTest [Runner: JUnit 3]
    test100 (0.000 s)
    test10000 (0.000 s)
    test1000 (0.000 s)
    test10 (0.000 s)
```

```
package arraySorter;

import RandomArray.*;

/**
 * The test class BubbleSortTest.
 *
 * @author  Hugh Osborne
 * @version October 2016
 */
public class QuickSortTest extends SortTester<Integer>
{
    private QuickSort<Integer> sorter = new QuickSort<Integer>();
    private RandomArray<Integer> generator = new RandomIntegerArray(1000000);

    /**
     * Default constructor for test class BubbleSortTest
     */
    public QuickSortTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    protected void setUp()
    {
    }

    /**
     * Tears down the test fixture.
     *
     * Called after every test case method.
     */
```



```
<terminated> QuickSortTimer [Java Application] C:\Program Files\Java\jre1.8.0_121\b
Average time to sort 1 elements was 0.062 milliseconds.
Average time to sort 2 elements was 0.002 milliseconds.
Average time to sort 3 elements was 0.004 milliseconds.
Average time to sort 4 elements was 0.002 milliseconds.
Average time to sort 5 elements was 0.002 milliseconds.
Average time to sort 6 elements was 0.006 milliseconds.
Average time to sort 7 elements was 0.006 milliseconds.
Average time to sort 8 elements was 0.003 milliseconds.
Average time to sort 9 elements was 0.003 milliseconds.
Average time to sort 10 elements was 0.003 milliseconds.
Average time to sort 20 elements was 0.003 milliseconds.
Average time to sort 30 elements was 0.004 milliseconds.
Average time to sort 40 elements was 0.007 milliseconds.
Average time to sort 50 elements was 0.008 milliseconds.
Average time to sort 60 elements was 0.012 milliseconds.
Average time to sort 70 elements was 0.072 milliseconds.
Average time to sort 80 elements was 0.010 milliseconds.
Average time to sort 90 elements was 0.012 milliseconds.
Average time to sort 100 elements was 0.013 milliseconds.
Average time to sort 200 elements was 0.030 milliseconds.
Average time to sort 300 elements was 0.049 milliseconds.
Average time to sort 400 elements was 0.063 milliseconds.
Average time to sort 500 elements was 0.082 milliseconds.
Average time to sort 600 elements was 0.101 milliseconds.
Average time to sort 700 elements was 0.120 milliseconds.
Average time to sort 800 elements was 0.141 milliseconds.
Average time to sort 900 elements was 0.158 milliseconds.
Average time to sort 1000 elements was 0.205 milliseconds.
Average time to sort 2000 elements was 0.394 milliseconds.
Average time to sort 3000 elements was 1.963 milliseconds.
Average time to sort 4000 elements was 2.806 milliseconds.
Average time to sort 5000 elements was 3.281 milliseconds.
Average time to sort 6000 elements was 0.613 milliseconds.
Average time to sort 7000 elements was 0.765 milliseconds.
Average time to sort 8000 elements was 0.906 milliseconds.
Average time to sort 9000 elements was 1.029 milliseconds.
Average time to sort 10000 elements was 1.127 milliseconds.
Average time to sort 20000 elements was 2.500 milliseconds.
Average time to sort 30000 elements was 4.001 milliseconds.
Average time to sort 40000 elements was 5.507 milliseconds.
Average time to sort 50000 elements was 7.119 milliseconds.
Average time to sort 60000 elements was 8.666 milliseconds.
Average time to sort 70000 elements was 9.936 milliseconds.
Average time to sort 80000 elements was 11.495 milliseconds.
Average time to sort 90000 elements was 15.201 milliseconds.
Average time to sort 100000 elements was 15.130 milliseconds.
Average time to sort 200000 elements was 33.382 milliseconds.
Average time to sort 300000 elements was 54.889 milliseconds.
Average time to sort 400000 elements was 73.444 milliseconds.
Average time to sort 500000 elements was 96.158 milliseconds.
```

SelectionSort

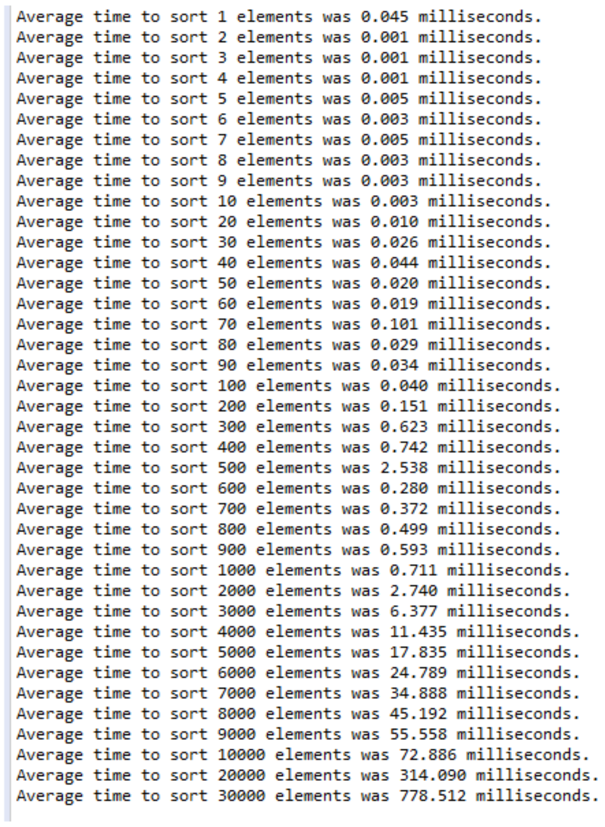

Workspace - Java - Week5/src/arraySorter/SelectionSortTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Test Field Assist Menu X2 Run Window Help
Package Explorer
JUnit
QuickSort.java
QuickSortTe...
SortTester.java
QuickSortTim...
Failure Trace

Finished after 0.124 seconds

Runs: 4/4 Errors: 0 Failures: 0

arraySorter.SelectionSortTest [Runner: JUn
test100 (0.001 s)
test10000 (0.000 s)
test1000 (0.000 s)
test10 (0.000 s)

```
package arraySorter;

import RandomArray.*;

public class SelectionSortTest extends SortTester<Integer>
{
    private SelectionSort<Integer> sorter = new SelectionSort<Intege
    private RandomArray<Integer> generator = new RandomIntegerArray

    /**
     * Default constructor for test class SelectionSortTest
     */
    public SelectionSortTest()
    {
    }

    /**
     * Sets up the test fixture.
     */
    protected void setUp()
    {
    }

    /**
     * Tears down the test fixture.
     */
    protected void tearDown()
    {
    }

    public void test10()
    {
        super.test(sorter,generator,10);
    }
```

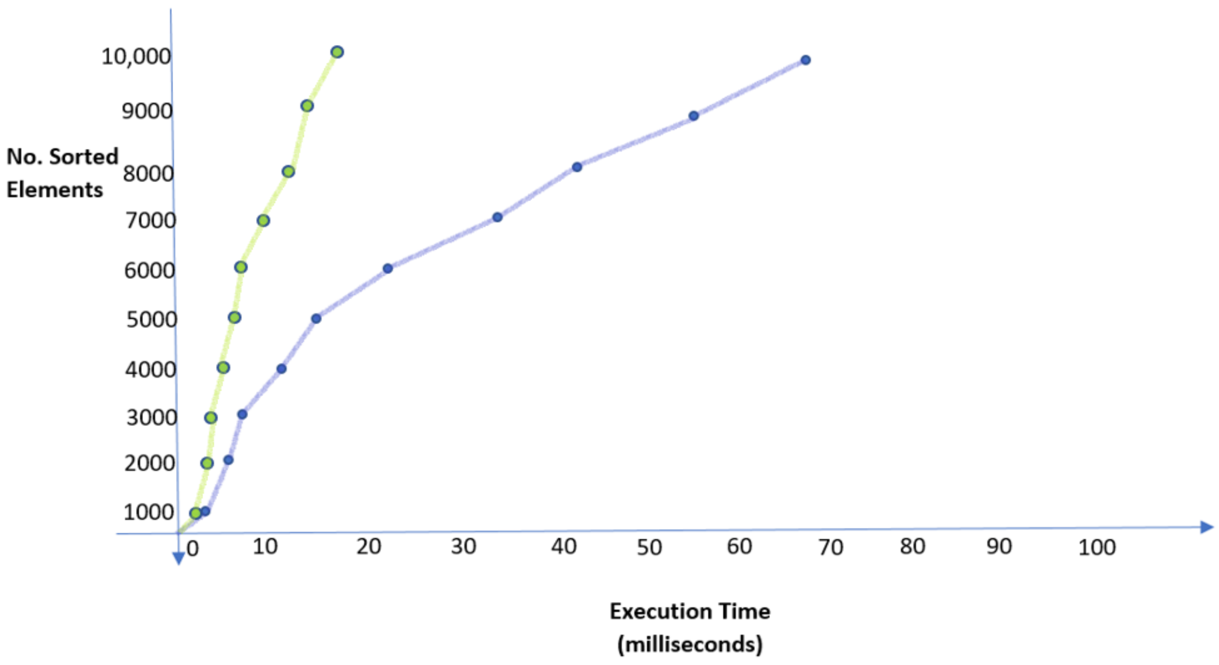


```
Average time to sort 1 elements was 0.045 milliseconds.
Average time to sort 2 elements was 0.001 milliseconds.
Average time to sort 3 elements was 0.001 milliseconds.
Average time to sort 4 elements was 0.001 milliseconds.
Average time to sort 5 elements was 0.005 milliseconds.
Average time to sort 6 elements was 0.003 milliseconds.
Average time to sort 7 elements was 0.005 milliseconds.
Average time to sort 8 elements was 0.003 milliseconds.
Average time to sort 9 elements was 0.003 milliseconds.
Average time to sort 10 elements was 0.003 milliseconds.
Average time to sort 20 elements was 0.010 milliseconds.
Average time to sort 30 elements was 0.026 milliseconds.
Average time to sort 40 elements was 0.044 milliseconds.
Average time to sort 50 elements was 0.020 milliseconds.
Average time to sort 60 elements was 0.019 milliseconds.
Average time to sort 70 elements was 0.101 milliseconds.
Average time to sort 80 elements was 0.029 milliseconds.
Average time to sort 90 elements was 0.034 milliseconds.
Average time to sort 100 elements was 0.040 milliseconds.
Average time to sort 200 elements was 0.151 milliseconds.
Average time to sort 300 elements was 0.623 milliseconds.
Average time to sort 400 elements was 0.742 milliseconds.
Average time to sort 500 elements was 2.538 milliseconds.
Average time to sort 600 elements was 0.280 milliseconds.
Average time to sort 700 elements was 0.372 milliseconds.
Average time to sort 800 elements was 0.499 milliseconds.
Average time to sort 900 elements was 0.593 milliseconds.
Average time to sort 1000 elements was 0.711 milliseconds.
Average time to sort 2000 elements was 2.740 milliseconds.
Average time to sort 3000 elements was 6.377 milliseconds.
Average time to sort 4000 elements was 11.435 milliseconds.
Average time to sort 5000 elements was 17.835 milliseconds.
Average time to sort 6000 elements was 24.789 milliseconds.
Average time to sort 7000 elements was 34.888 milliseconds.
Average time to sort 8000 elements was 45.192 milliseconds.
Average time to sort 9000 elements was 55.558 milliseconds.
Average time to sort 10000 elements was 72.886 milliseconds.
Average time to sort 20000 elements was 314.090 milliseconds.
Average time to sort 30000 elements was 778.512 milliseconds.
```

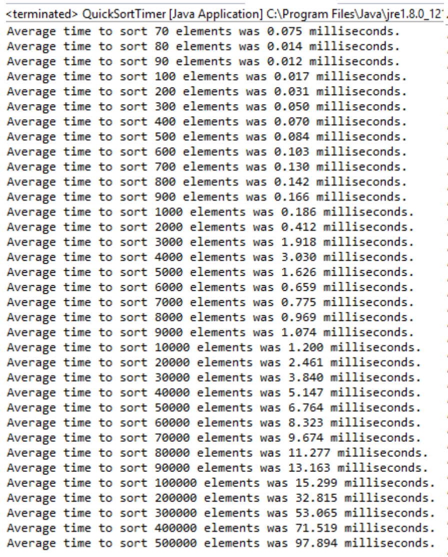
Graph

Key:

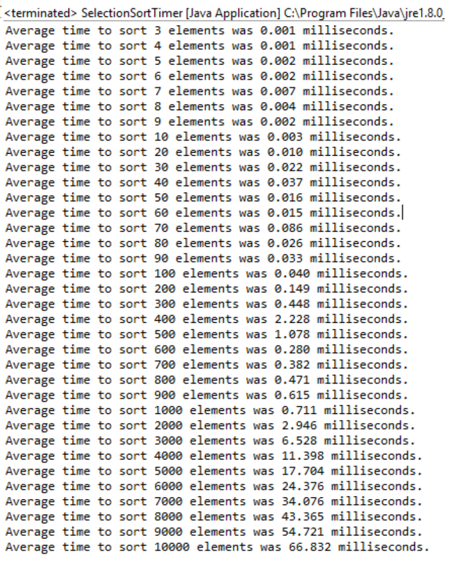
Green = Quicksort
Blue = SelectionSort



Output



```
<terminated> QuickSortTimer [Java Application] C:\Program Files\Java\jre1.8.0_12
Average time to sort 70 elements was 0.075 milliseconds.
Average time to sort 80 elements was 0.014 milliseconds.
Average time to sort 90 elements was 0.012 milliseconds.
Average time to sort 100 elements was 0.017 milliseconds.
Average time to sort 200 elements was 0.031 milliseconds.
Average time to sort 300 elements was 0.050 milliseconds.
Average time to sort 400 elements was 0.070 milliseconds.
Average time to sort 500 elements was 0.084 milliseconds.
Average time to sort 600 elements was 0.103 milliseconds.
Average time to sort 700 elements was 0.130 milliseconds.
Average time to sort 800 elements was 0.142 milliseconds.
Average time to sort 900 elements was 0.166 milliseconds.
Average time to sort 1000 elements was 0.186 milliseconds.
Average time to sort 2000 elements was 0.412 milliseconds.
Average time to sort 3000 elements was 1.918 milliseconds.
Average time to sort 4000 elements was 3.030 milliseconds.
Average time to sort 5000 elements was 1.626 milliseconds.
Average time to sort 6000 elements was 0.659 milliseconds.
Average time to sort 7000 elements was 0.775 milliseconds.
Average time to sort 8000 elements was 0.969 milliseconds.
Average time to sort 9000 elements was 1.074 milliseconds.
Average time to sort 10000 elements was 1.200 milliseconds.
Average time to sort 20000 elements was 2.461 milliseconds.
Average time to sort 30000 elements was 3.840 milliseconds.
Average time to sort 40000 elements was 5.147 milliseconds.
Average time to sort 50000 elements was 6.764 milliseconds.
Average time to sort 60000 elements was 8.323 milliseconds.
Average time to sort 70000 elements was 9.674 milliseconds.
Average time to sort 80000 elements was 11.277 milliseconds.
Average time to sort 90000 elements was 13.163 milliseconds.
Average time to sort 100000 elements was 15.299 milliseconds.
Average time to sort 200000 elements was 32.815 milliseconds.
Average time to sort 300000 elements was 53.065 milliseconds.
Average time to sort 400000 elements was 71.519 milliseconds.
Average time to sort 500000 elements was 97.894 milliseconds.
```



```
<terminated> SelectionSortTimer [Java Application] C:\Program Files\Java\jre1.8.0
Average time to sort 3 elements was 0.001 milliseconds.
Average time to sort 4 elements was 0.001 milliseconds.
Average time to sort 5 elements was 0.002 milliseconds.
Average time to sort 6 elements was 0.002 milliseconds.
Average time to sort 7 elements was 0.007 milliseconds.
Average time to sort 8 elements was 0.004 milliseconds.
Average time to sort 9 elements was 0.002 milliseconds.
Average time to sort 10 elements was 0.003 milliseconds.
Average time to sort 20 elements was 0.010 milliseconds.
Average time to sort 30 elements was 0.022 milliseconds.
Average time to sort 40 elements was 0.037 milliseconds.
Average time to sort 50 elements was 0.016 milliseconds.
Average time to sort 60 elements was 0.015 milliseconds.
Average time to sort 70 elements was 0.086 milliseconds.
Average time to sort 80 elements was 0.026 milliseconds.
Average time to sort 90 elements was 0.033 milliseconds.
Average time to sort 100 elements was 0.040 milliseconds.
Average time to sort 200 elements was 0.149 milliseconds.
Average time to sort 300 elements was 0.448 milliseconds.
Average time to sort 400 elements was 2.228 milliseconds.
Average time to sort 500 elements was 1.078 milliseconds.
Average time to sort 600 elements was 0.280 milliseconds.
Average time to sort 700 elements was 0.382 milliseconds.
Average time to sort 800 elements was 0.471 milliseconds.
Average time to sort 900 elements was 0.615 milliseconds.
Average time to sort 1000 elements was 0.711 milliseconds.
Average time to sort 2000 elements was 2.946 milliseconds.
Average time to sort 3000 elements was 6.528 milliseconds.
Average time to sort 4000 elements was 11.398 milliseconds.
Average time to sort 5000 elements was 17.704 milliseconds.
Average time to sort 6000 elements was 24.376 milliseconds.
Average time to sort 7000 elements was 34.076 milliseconds.
Average time to sort 8000 elements was 43.365 milliseconds.
Average time to sort 9000 elements was 54.721 milliseconds.
Average time to sort 10000 elements was 66.832 milliseconds.
```

Additional Material

Understanding Big O Notation

This week I have taken It upon myself to try to understand Big O Notation more clearly. Below are the results of my tests to further understand how Big O Notation works, and provide myself with examples of which I can look back at in the future.

Big O notation gives us a feeling of the upper limit of how long execution will take "n" being the "volume of data";

For instance 0(1) has the same runtime regardless of n , this means no matter how big the problem is the time to execute will always be the same, much like this order of 0(N) grows in direct proportion to the volume of data, if we have a volume of data 1000 times our original volume, it will take 1000 times longer I.e. if we have a program that takes 10 seconds to execute it will take 10,000. We categories Big O Notation as

// most desirable (fast execution)
O(log n) logarithmic
O(n) linear
O(n log n)

// not as efficient, but still acceptable time-scale for execution
O(n^2) quadratic
O(n^3)

// not possible to calculate large problems
O(m^n) exponential

Although some problems may be solved within a acceptable amount of time for the answer needed, the algorithms we want are the most efficient, this means we do not want O(m^n). Exponential problems grow very quickly, these can be useful (e.g. in Cryptography, where we want to make items computationally difficult to calculate to prevent the key being cracked) however, usually this would be undesirable as a problem of size 10000 or more, may never be solved in a exponential problem

```
public class BigONotation {

        private int[] theArray;
        private int arraySize, itemsInArray;
        static long startTime, endTime;

        public BigONotation() {
                this.itemsInArray = 0;
        }

        public static void main(String[] args) {}

        // O(1) - Order Of 1
        public void addItemToArray(int newItem) {
                theArray[itemsInArray++] = newItem; //O(1) has the same runtime, regardless of 
ArraySize
        }

        // O(N) - Order Of N - time grows in direct proportion to amount of data
        public void linearSearch(int value) {

                boolean valueInArray = false;
                String indexWithValue = "";

                startTime = System.currentTimeMillis();

                for(int i = 0; i < arraySize; i++) {
                        if(theArray[i] == value) {
                                valueInArray = true;
                                indexWithValue += i + "";
                        }
                }

                System.out.println("Value Found: " + valueInArray);
                endTime = System.currentTimeMillis();
                System.out.println("Linear Search Took: " + (endTime - startTime));
        }

        // O(N^2)  time proportional to number of items squared (e.g. BubbleSort)
        public void BubbleSort() {
                startTime = System.currentTimeMillis();

                for(int i = arraySize -1; i > 1; i-- ) {
                        for(int j = 0; j < i; j++) {
                                if(theArray[j] < theArray[j+1]) {
                                        swapValues(j, j+1);
                                }
                        }

                        endTime = System.currentTimeMillis();
                }
        }

        /*
         * O(log N) data being used, decreased roughly by 50% on each iteration.
         * log N algorithms are efficient as increasing the amount
         *  of data has little or no effect.
         */
        public void binarySearch(int value) {
                startTime = System.currentTimeMillis();

                int lowIndex = 0;
                int highIndex = arraySize -1;

                int timesIterated = 0;

                while(lowIndex <= highIndex) {
                        int middleIndex = (highIndex + lowIndex) /2;

                        if(theArray[middleIndex] < value) {
```

```
                        lowIndex = middleIndex + 1;
                }
                else if(theArray[middleIndex] > value) {
                        highIndex = middleIndex -1;
                }
                else {
                        System.out.println("Found Match In Index " + middleIndex);
                }

                timesIterated++;

        }
        endTime = System.currentTimeMillis();
}

/*
 *  O(N log N) numberOfComparisons = log n! (! = factorial)
 *  i.e. Comparisons = log n + log(n-1)+....... + log(1)
 *  Comparisons = n log n
 */

public void quickSort(int left, int right) {
        if(right - left <= 0) {
                return;
        }
        else {
                int pivot = theArray[right];
                int pivotLocation = partitionArray(left, right, pivot);
                quickSort(left, pivotLocation -1);
                quickSort(pivotLocation +1, right);
        }
}

public int partitionArray(int left, int right, int pivot) {
        int leftPointer = left -1;
        int rightPointer = right;

        while(true) {
                while(theArray[++leftPointer] < pivot);
                while(rightPointer > 0 && theArray[--rightPointer] > pivot);

                if(leftPointer >= rightPointer) {
                        break;
                }
                else {
                        swapValues(leftPointer, right);
                }
        }
        return leftPointer;
}

public void swapValues(int j, int i) {

        int temp = j;
        j = i;
        i = temp;

}

public void genRandomArray() {

        for(int i = 0; i < arraySize; i++) {

        }

        itemsInArray = arraySize -1;
}
}
```

Week6

1) Implement the List<T> interface, using singly linked lists.

SingleLinkedList.java

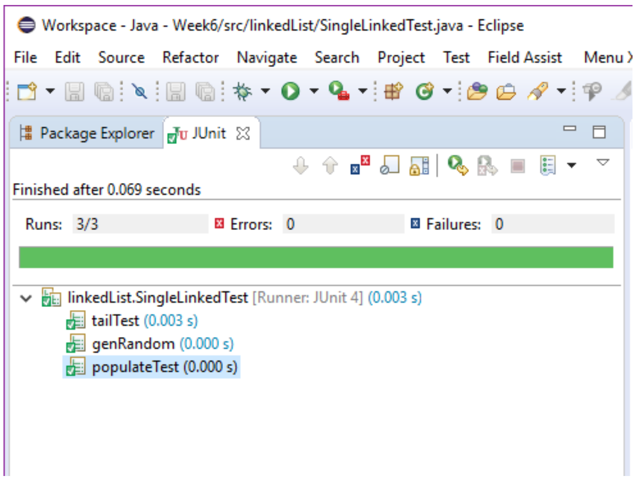
```
package linkedList;

import java.lang.reflect.Array;

public class SingleLinkedList<T> implements List {

        private Node<T> head;
        private Node<T> tail;

        private int currentLength;

        /**
         * Construct a empty list
         */
        public SingleLinkedList() {
                this.head = null;
                this.tail = null;
                this.currentLength = 0;
        }

        /**
         * returns the current size of SingleLinkedList
         * @return length of SingleLinkedList
         */
        public int size() {
                return this.currentLength;
        }

        /**
         * Check if next list is empty
         * @return true if empty
         */
        @Override
        public boolean isEmpty() {
                return (this.currentLength == 0);
        }

        /**
         * adds a new node to the list
         * @param node the node to be added
         * @return always returns true
         */
        public boolean add(Object node) {
                if(isEmpty()) {
                        this.head = new Node(node, null);
                        this.tail = head;
                }
                else {
                        tail.setNext(new Node(node, null));
                        tail = tail.getNext();
                }
                currentLength++;
                return true;
        }

    /**
     * Add a new object to the list at position index
     * shifts elements currently at index position over
     * @throw ListAccessError if index is an invalid index
     * @throws IndexOutOfBoundException if index > size() or index < 0
     */
        @Override
        public void add(int index, Object value) throws ListAccessError {
                if(index < 0 || index > size()) {
                        throw new IndexOutOfBoundsException();
                }
                if(index == size()) {
                        add(value); //appends to end of list
                        return;
                }
                if(index == 0) {
                        this.head = new Node(value, head);
                }
```

```
		else {
			Node<T> temp = head;
			for(int i = 0; i < index-1; i++) {
				temp = temp.getNext(); //temp is the node before the index
			}
			temp.setNext(new Node(value, temp.getNext()));
		}
		currentLength++;
	}

	/**
	 * removes all elements from the list
	 */
	public void clear() {
		head = null;
		tail = null;
		currentLength = 0;
	}

	@Override
	public Object remove(int index) throws ListAccessError {
		// TODO Auto-generated method stub
		return null;
	}

	/**
	 * returns the element at position index
	 * @return element at position index
	 * @throw IndexOutOfBoundsException if index out of range (index < 0 || index >=
size())
	 * @throws ListAccessError if index is invalid
	 */
	@Override
	public Object get(int index) throws ListAccessError {
		if(index < 0 || index >= size()) {
			throw new IndexOutOfBoundsException();
		}
		if(index < (size()-1)) {
			Node<T> temp = head;
			for(int i = 0; i < index; i++) {
				temp = temp.getNext();
			}
			return temp.getValue();
		}
		return tail.getValue();
	}
} // end of Class SingleLinkedList
```

SingleLinkedListTest.java

```
package linkedList;

import static org.junit.Assert.*;

import java.util.Random;

import org.junit.Test;

public class SingleLinkedTest {

	private SingleLinkedList<String> list = new SingleLinkedList<String>();
	private int listSize = list.size();
	private int low = 1;
	private int random1, random2, random3, random4;

	private Object tailObject;

	/**
	 * attempts to create an initial population within the list
	 * @throws ListAccessError if index is invalid
	 */
	@Test
	public void populateTest() throws ListAccessError {
		int i = 0;
		while(i != this.listSize) {
			if(this.listSize < 0) {
				System.out.println("Please set list size");
				break;
			}
			else {
				try {
					list.add(this.random1, "A");
					list.add(this.random2, "B");
					list.add(this.random3, "C");
				    list.add(this.random4, "D");
				}
				catch(Exception e) {
					System.out.println("Something went wrong");
				}
				if(list.get(random1) == "A" && list.get(random3) == "C") {
					//do nothing
				}
				else {
					fail();
				}
			}
		}
	}

	/**
	 * generates a random integer between low and listSize
	 * @param low the smallest possible index to generate at
	 * @return
	 * @throws ListAccessError if invalid index
	 */
	@Test
	public void genRandom() throws ListAccessError {
		Random random = new Random(list.size()-1);

		for(int i = random.nextInt(); i < list.size(); i++) {
			list.add(list.get(i));
		}
	}

	/**
	 * adds a new element as the tail of the list, tests to ensure
	 * the object is added to the end
	 * @throws ListAccessError if invalid index
	 */
	@Test
	public void tailTest() throws ListAccessError {
		populateTest();
		list.add("tail");

		for(int i = 0; i < list.size(); i++) {
			if(list.get(list.size()-1) == "tail") {
				//do nothing
			}
			else {
				fail();
			}
		}
	}
} // end of Class SingleLinkedTest
```

Output


Workspace - Java - Week6/src/linkedList/SingleLinkedTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Test Field Assist Menu
Package Explorer
JUnit

```
Finished after 0.069 seconds

Runs: 3/3    Errors: 0    Failures: 0

linkedList.SingleLinkedTest [Runner: JUnit 4] (0.003 s)
    tailTest (0.003 s)
    genRandom (0.000 s)
    populateTest (0.000 s)
```

Week7

1) Complete the implementation of the binary tree class shown in the lecture.

Binary Tree

Binary trees are depicted by using one root value at the top of the tree, the larger values are placed on the left side, and the smaller ones on the right.

BinaryTree.java

```
package binaryTree;

/**
 *@author Luke McCann - U1364096
 * BinaryTree - the main functions for the binary tree
 * insert a value, inordertraversal, postordertraversal and preordertraversal
 */
public class BinaryTree<T extends Comparable<T>> implements BTree<T> {

        private TreeNode<T> root;

        /*
         * creates a new tree, compares inserted
         * number to the current top node and is placed
         * on the corresponding side of the tree
         * @param value - value to be inserted
         */
        @Override
        public void insert(T value) {
                if (root == null) {
                        root = new TreeNode<T>(value);
                }
                else if (value.compareTo(root.value()) < 0) {
                        System.out.println("Left tree: ");
                        root.left().insert(value);
                        }
                else {
                        System.out.println("Right tree: ");
                        root.right().insert(value);
                        }
        }

        /*
         * performs a inordertraversal of the tree
         */
        public void inOrderTraverse(BTree<T> node) {
                if(node != null) {
                        preOrderTraverse(node.left());

                        System.out.println(node);

                        preOrderTraverse(node.right());
                }
        }

        /*
         * performs a preordertraversal of the tree
         */
        public void preOrderTraverse(BTree<T> node) {
                if(node != null) {
                        System.out.println(node);

                        preOrderTraverse(node.left());
                        preOrderTraverse(node.right());
                }
        }

        /*
```

```
	 * performs a postordertraversal of the tree
	 */
	public void postOrderTraverse(BTree<T> node) {
		preOrderTraverse(node.left());
		preOrderTraverse(node.right());

		System.out.println(node);
	}


	/*
	 * returns and prints root value
	 */
	@Override
	public T value() {
		System.out.println("Root is: " + root.value());
		return root.value();
	}

	/*
	 * prints and returns value going left
	 */
	@Override
	public BTree<T> left() {
		System.out.println("Left: " + root.left());
		return root.left();
	}

	@Override
	public BTree<T> right() {
		System.out.println("Right: " + root.right());
		return root.left();
	}

} //End of class Binarytree
```

BTree.java

```
package binaryTree;

/**
 *
 * @author Luke McCann - U1364096
 * Binary tree interface, abstract methods to be implemented
 * @param <T>
 */
public interface BTree<T extends Comparable<T>> {
	public void insert(T value);

	public T value();

	public BTree<T> left();

	public BTree<T> right();

}//End of class BTree
```

TreeNode.java

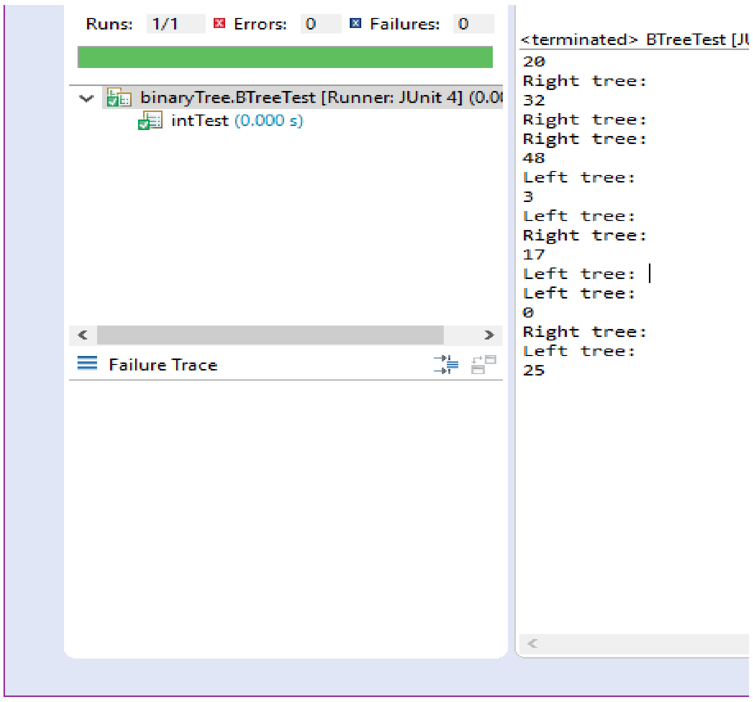
```
package binaryTree;

public class TreeNode<T extends Comparable<T>> {

	T value;
	BTree<T> left, right;

	public TreeNode(T value) {
		this.value = value;
		System.out.println(value());
		this.left = new BinaryTree<T>();
		this.right = new BinaryTree<T>();
```

```
        }

        public T value() {
                return value;
        }

        public void setValue(T value) {
                this.value = value;
        }

        public BTree<T> left() {
                return left;
        }

        public BTree<T> right() {
                return right;
        }

        public String toString() {
                return "Node Value: " + value;
        }

} //End of class TreeNode
```

BTreeTest.java

```
package binaryTree;

import static org.junit.Assert.*;

import org.junit.Test;

public class BTreeTest {

        @Test
        public void intTest() {
                BinaryTree<Integer> tree = new BinaryTree<Integer>();

                tree.insert(20);
                tree.insert(32);
                tree.insert(48);
                tree.insert(3);
                tree.insert(17);
                tree.insert(0);
                tree.insert(25);

        }

}
```

Output


Runs: 1/1 Errors: 0 Failures: 0

binaryTree.BTreeTest [Runner: JUnit 4] (0.0
intTest (0.000 s)

Failure Trace

```
<terminated> BTreeTest [JU
20
Right tree:
32
Right tree:
Right tree:
48
Left tree:
3
Left tree:
Right tree:
17
Left tree: |
Left tree:
0
Right tree:
Left tree:
25
```

Week8/9

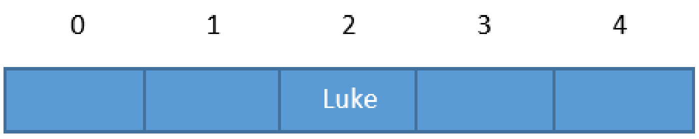
Hashtables

Explanation

Hashtables usually consist of a set of values and some form of a key. Hashtables are often used to implement associative arrays, these can map keys to a value. The heart of the Hashtable is basically an array structure;

Key: name

Value: student_number



The first step of the Hashtable is the Hash function. This function takes in a key as a value and finds the index number, of the value associated with this key, where it is stored within the array I.e.;

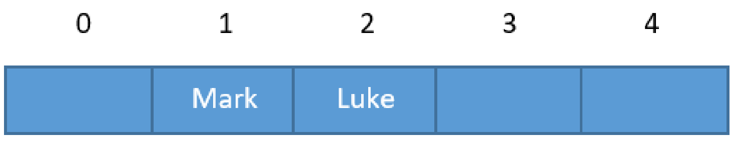
Hash(key)   returns: index

Hash(Luke) returns: 2

This would mean that the value associated with the name Luke is at index 2 in the array, unless the position in the array changes, this should not change, every time we enter the key “Luke” the value returned should be 2, if we were to add another person to the array we the rules would be the same, for example;

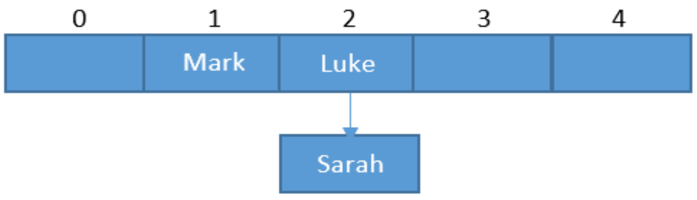
Hash(Mark) return: 1

Would mean that Mark's student number is stored in index 1 and Luke's is still stored in index 2.



If we wanted to add another person whose hash value is 2 we end up with a collision as there is already a value stored in index 2, in order to avoid this collision we can create a Linked List from one of the indexes in order to store the persons data in the indexed position, this technique is known as chaining, each of these indexes links on to the last stored at that index creating a form of chain. As an example we can add another student to our array;

Hash(Sarah) return: 2



As Sarah has her student number stored at the same index value as a previously existing student her information is linked from Luke's number at position 2, in this method anytime we end up with a collision the person who was last added is simply chained to that same index position. The nice thing about this form of storage is that if we end up with a huge number of students within the array we can easily find which index their data is stored in simply by using Hash(person) to return the index value. From this we then know we do not need to look in any other index for this information, for example if the hash function returns an index value of 3 we know that the person we are looking for has their information stored somewhere at index position 3, we could then look in the list (or chain) to find exactly where that persons information is located.

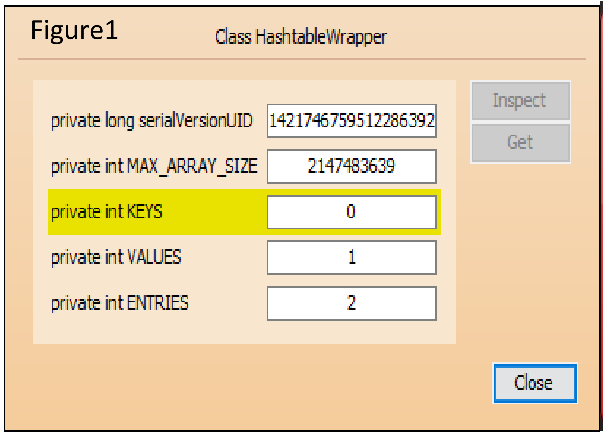
Hashtables offer fast insertion and searching, they are however limited in size and hard to order, they also have time constant access meaning that they time taken to search is directly proportionate to the number of elements within the array.

In the tutorial example an instance of HashtableWrapper is created and initialized with an initial size value of 5;

Initialized Example (BlueJay)

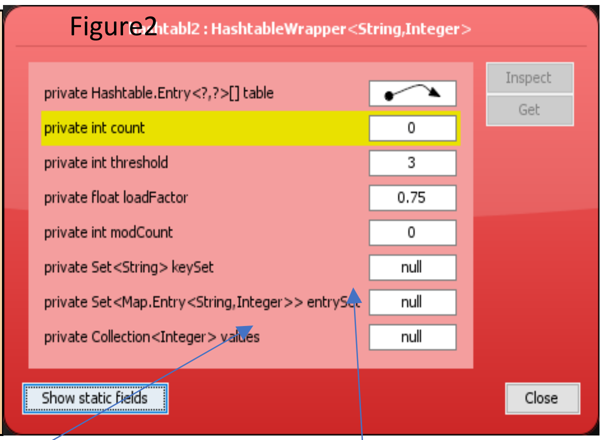
Size 5

Figure1


Class HashtableWrapper

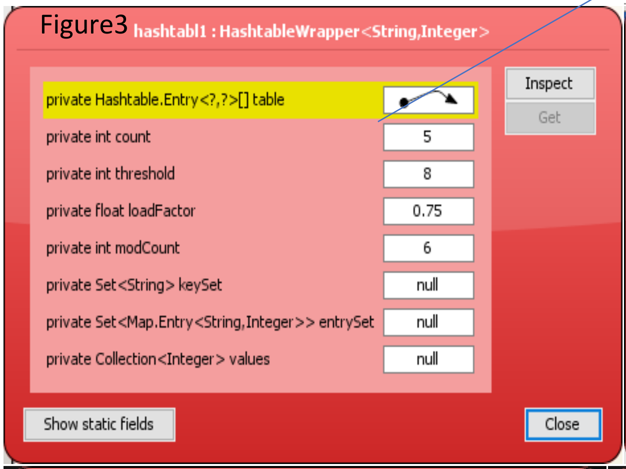
private long serialVersionUID	1421746759512286392
private int MAX_ARRAY_SIZE	2147483639
private int KEYS	0
private int VALUES	1
private int ENTRIES	2

Figure2


htabl2 : HashtableWrapper<String,Integer>

private Hashtable.Entry<?,?>[] table	→
private int count	0
private int threshold	3
private float loadFactor	0.75
private int modCount	0
private Set<String> keySet	null
private Set<Map.Entry<String,Integer>> entrySet	null
private Collection<Integer> values	null

Figure3


hashtabl1 : HashtableWrapper<String,Integer>

private Hashtable.Entry<?,?>[] table	→
private int count	5
private int threshold	8
private float loadFactor	0.75
private int modCount	6
private Set<String> keySet	null
private Set<Map.Entry<String,Integer>> entrySet	null
private Collection<Integer> values	null

Figure4

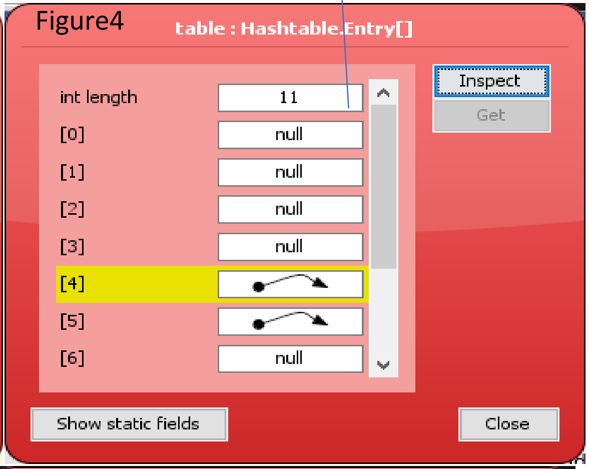
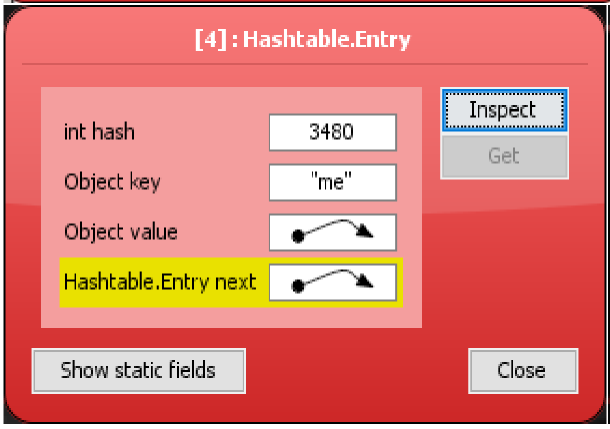
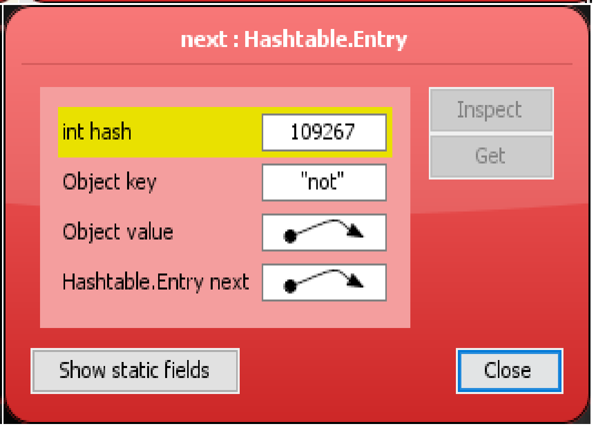


table : Hashtable.Entry[]

int length	11
[0]	null
[1]	null
[2]	null
[3]	null
[4]	→
[5]	→
[6]	null


[4] : Hashtable.Entry

int hash	3480
Object key	"me"
Object value	→
Hashtable.Entry next	→


next : Hashtable.Entry

int hash	109267
Object key	"not"
Object value	→
Hashtable.Entry next	→

As our initial array size is set to 5, the threshold as of this moment is 8, this is because the Threshold is the maximum number of elements which can be held in the Hashtable before the size value of the table increases. In hashing the load factor determines how full the hash table is before it needs to increase the arrays maximum capacity, it is set initially at 0.75 and after inspection and testing it can be determined this means that when the table is 75% full the table will increase, hence when you add 3 items into a 5 item array you end up with a threshold of 8, in other words when the Hashtable is 75% full a new hash map is created, expanding the storage capacity.

When we add the values:

– ("is",69) – ("dead",0) – ("but",999) – ("not",-42) – ("me!",-1)

to the array, we have some strange behaviour. This is observed as a seemingly “random” storage method at first glance. The reasoning behind this is that the values we are inserting contain
values of addresses at which do not yet exist (our array only has a size of 5). To work out where these elements will be placed we take the hash code (e.g. 109267) modulo of 5 (this leaves us with a remainder of 4) where our “not”,-42 object is correctly mapped to [4].

Week10

DepthFirstTraversal.java (Working: Revisited Version at end of logbook)

<Deprecated>

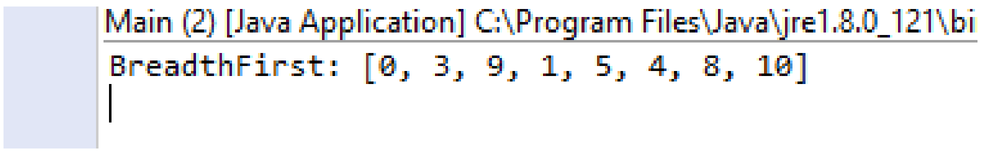
```java
package graph;

import java.util.ArrayList;
import java.util.HashSet;

/**
 * An implementation of generic DepthFirstTraversal using Traversal interface
 *
 * @author U1364096 - Luke McCann
 * @version January 2017
 */

import java.util.List;
import java.util.Set;

public class DepthFirstTraversal<T> extends AdjacencyGraph<T> implements Traversal<T> {

	private List<T> traversal = new ArrayList<T>();
	private Set<T> visited = new HashSet<T>();

	/**
	 * visits each node in <tt>list (traversal)</tt>, returns the traversal as an
	 * (ArrayList<T>)
	 * @return the <tt>list (traversal)</tt>
	 */
	@Override
	public List<T> traverse() throws GraphError {
		while(nextUnvisited() != null) {
			visitNode(nextUnvisited());
		}
		return (ArrayList<T>) traversal;
	}

	/**
	 * looks for the next <tt>node</tt> in the list, if it has not been visited returns
	 * the <tt>node</tt>
	 * @return the next <tt>node</tt> to be visited
	 */
	protected T nextUnvisited() {
		for(T node : getNodes()) {
			if(!visited.contains(node)) {
				return node;
			}
		}
		return null;
	}

	/**
	 * Adds <tt>node</tt> to visited and traverses, checks if <tt>node(next)</tt>
	 * has been visited, and recursively adds unvisited nodes
	 * @param <tt>node</tt> the <tt>node</tt> to begin traversal at
	 * @throws GraphError if <tt>node</tt> does not exist
	 */
	public void visitNode(T node) throws GraphError {
		visited.add(node);
		traversal.add(node);
		for(T next : neighbours(node)) {
			if(!visited.contains(next)) {
				visitNode(next);
			}
		}
	}
} // end of Class DepthFirstTraversal
```

DepthFirstTest.java

```
package graph;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

import java.util.ArrayList;
import java.util.Random;

import org.junit.Test;

import junit.framework.~~Assert~~;

public class DepthFirstTest<T> {

	private int lowVal;
	private int highVal;
	private ArrayList<Object> linked = new ArrayList<>();

	DepthFirstTraversal<Integer> dftInt = new DepthFirstTraversal<Integer>();

	public DepthFirstTest() {
		this.lowVal = 0;
		this.highVal = 10;
	}


	/**
	 * manual graph population
	 * @throws GraphError if node already exists
	 */
	@Test
	public void manualTest() throws GraphError {
		System.out.println("Manual Test: ");
		dftInt.add(0);
		dftInt.add(2);
		dftInt.add(4);
		dftInt.add(3);
		dftInt.add(5);
		dftInt.add(1);
		dftInt.add(1,2);
		dftInt.add(3,5);
		dftInt.add(2,3);
		dftInt.add(2,1);
		dftInt.add(2,4);
		dftInt.add(2,5);
		dftInt.add(4,5);
		dftInt.add(5,4);

		Object output = dftInt.traverse();
		System.out.println(dftInt.getTraversalList());
		System.out.println("");

		assertEquals(dftInt.getTraversalList(),output);
	}

	/**
	 * populates the graph in order and traverses
	 * if the output is not as expected the test fails
	 * @throws GraphError if <tt>node</tt> already exists
	 */
	@Test
	public void inOrdersetup() throws GraphError {
		System.out.println("In Order Setup: ");
		inOrderPopulate();
		Object output = traversalTest();
		linkedNodesTest();
		System.out.println("");

		assertEquals(dftInt.getTraversalList(),output);
	}

	/**
 * populates the graph with randomisd <tt>nodes</tt> between low and high+1
 * @throws GraphError if the node already exists
 */
@Test
public void randomOrderetup() throws GraphError {
        System.out.println("Random Order Setup: ");
        randomOrderPopulate();
        Object output = traversalTest();

        assertEquals(dftInt.getTraversalList(),output);
}

/**
 * this test should always fail
 * attempts to add a random node between 1 and 10 to a
 * already populated Graph
 * @throws GraphError if node already exists
 */
@Test
public void nodeExistsTest() throws GraphError {
        inOrderPopulate();
        dftInt.add(genRandom());
}

/**
 * this test should always fail
 * attempts to link missing <tt>nodes</tt>
 * @throws GraphError if <tt>nodes</tt> does not exist
 */
@Test
public void nodeMissingTest() throws GraphError {
        linkNodesTest();
}

/**
 * attempts to traverse <tt>dfsInt</tt> graph
 * @return the traversal
 * @throws GraphError if the <tt>node</tt> does not exist
 */
public Object traversalTest() throws GraphError {
        try {
                dftInt.traverse();
                System.out.println("Traversal Test: "
                            +dftInt.getTraversalList());
        }
        catch(GraphError e) {
                e.printStackTrace();
        }
        return dftInt.traverse();
}

/**
 * tries to link two nodes
 * prints stack trace if unable to
 */
public void linkedNodesTest() {
        try {
                linkNodesTest();
        }
        catch(GraphError e) {
                e.printStackTrace();
        }
}

/**
 * populates the adjacency graph with values 1 to 10
 * @throws GraphError if the <tt>node</tt> already exists
 */
public void inOrderPopulate() throws GraphError {
        for(int i = 0; i < highVal+1; i++) {
                dftInt.add(i);
        }
}

/**
 * creates a population of random values from low to high
 * @throws GraphError if node already exists
```

```
         */
        public void randomOrderPopulate() throws GraphError {
                for(int i = 0; i < highVal+1; i++) {
                        int random = genRandom();
                        if(!dftInt.contains(random)) {
                                dftInt.add(random);
                        }
                        else {
                                random = genRandom();
                        }
                }
        }

    /**
     * links two random <tt>nodes</tt> in the graph
     * @throws GraphError if the link already exists
     */
    public void linkNodesTest() throws GraphError {
        if(!linked.contains(genRandom())) {
                int randomStart = genRandom();
            int randomEnd = genRandom();
                dftInt.add(randomStart, randomEnd);
                linked.add(randomStart);
                linked.add(randomEnd);
        }
        else {
                //do nothing
        }
        System.out.println("Random Nodes Linked: "
                           + linked);
     }

        /**
         * returns a random value between lowVal and highVal
         * @return random value between lowVal and highVal
         */
    public int genRandom() {
        Random random = new Random();
        return (random.nextInt(highVal - lowVal) + lowVal);
    }
} // end of Class DepthFirstTest
```

Outputs



```
Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_121\bi
BreadthFirst: [0, 3, 9, 1, 5, 4, 8, 10]
```

Week11

ReferenceSort.java

```
package graph;

import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;

public class ReferenceSort<T> extends AdjacencyGraph<T> implements TopologicalSort<T>{

        private List<T> sort = new ArrayList<T>();
        private HashMap <T, Integer> hash = new HashMap <T,Integer>();

        @Override
        public List<T> sort() throws GraphError {
                return sort;
        }

        /**
         * does a reference count topological sort on AdjacencyGraph
         * while the next none listed node != null
         * @throws GraphError
         */
        private void doSort() throws GraphError {
                T node;
                while((node = nextNoRefNode()) != null) {
                        visitNode(node);
                }
        }

        /**
         * adds the node to "sort"
         * @param node the <tt>node</tt> to visit
         * @throws GraphError
         */
        private void visitNode(T node) throws GraphError {
                sort.add(node);
                for(T successor: neighbours(node)) {
                    decreaseRefCount(successor);
                }
        }

        /**
         * initialises the reference count for
         * all nodes incrementing them.
         * @throws GraphError if node already exists
         */
        private void setUpRefCount() throws GraphError {
                for(T node: getNodes()) {
                        for(T successor : neighbours(node)) {
                                increaseRefCount(successor);
                        }
                }
        }

        /**
         * increases the reference count
         * @param successor the node to visit next
         */
        private void increaseRefCount(T successor) {
                if(hash.get(successor) != null) {
                        hash.put(successor, hash.get(successor)+1);
                }
        }

        /**
         * decreases the reference count
         * @param successor the node to visit next
         */
        private void decreaseRefCount(T successor) {
```

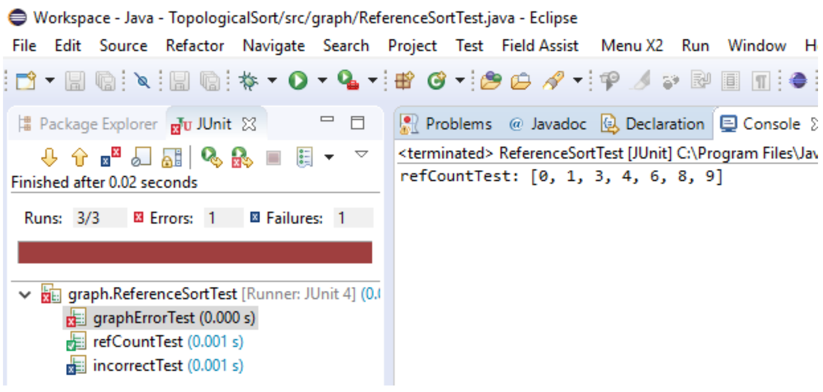
```
			if(hash.get(successor) != null) {
				hash.put(successor, hash.get(successor)-1);
			}
		}

		/**
		 * iterates through the nodes
		 * @return node - if the node
		 * is equal to 0 and is not contained in sort
		 * the node is returned
		 */
		private T nextNoRefNode() {
			for(T node : getNodes()) {
				if(hash.get(node) == 0 && !sort.contains(node)) {
					return node;
				}
			}
			return null;
		}
} // end of Class ReferenceSort
```

ReferenceSortTest.java

```
package graph;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import org.junit.Test;

public class ReferenceSortTest {

		/**
		 * initial test for ReferenceSort
		 * @throws GraphError if nodes do not exist
		 */
		@Test
		public void refCountTest() throws GraphError {
			ReferenceSort<Integer> x = new ReferenceSort<Integer>();
			List<Integer> sort = new ArrayList<>();

			// initialise nodes
			x.add(0); x.add(1); x.add(2); x.add(3);
			x.add(4); x.add(5); x.add(6); x.add(7);
			x.add(8); x.add(9);

			// connect nodes
			x.add(0, 5); x.add(9, 2); x.add(3,7);
			x.add(6,2); x.add(5,2);

			x.doSort();

			sort.add(0); sort.add(1); sort.add(3);
			sort.add(4); sort.add(6); sort.add(8);
			sort.add(9);

			assertEquals(sort, x.sort());
			System.out.println("refCountTest: " + sort);
		}

		/**
		 * tests that the test fails if results are incorrect
		 * @throws GraphError if node exists
		 */
		@Test
		public void incorrectTest() throws GraphError {
			ReferenceSort<Integer> x = new ReferenceSort<Integer>();
			List<Integer> sort = new ArrayList<>();

			// initialise nodes
			x.add(0); x.add(1); x.add(2); x.add(3);
```

```
                x.add(4); x.add(5); x.add(6); x.add(7);
                x.add(8); x.add(9);

                // connect nodes
                x.add(0, 5); x.add(9, 2); x.add(3,7);
                x.add(6,2); x.add(5,2);

                x.doSort();

                sort.add(1); sort.add(3); sort.add(6);
                sort.add(4); sort.add(5); sort.add(2);
                sort.add(9);

                assertEquals(sort, x.sort());
                System.out.println("Incorrect Test: " + sort);
        }

        /**
         * provides non existing nodes
         * @throws GraphError Always
         */
        @Test
        public void graphErrorTest() throws GraphError {
                ReferenceSort<Integer> x = new ReferenceSort<Integer>();
                List<Integer> sort = new ArrayList<>();

                // initialise nodes
                x.add(1); x.add(2); x.add(3);
                x.add(4); x.add(5); x.add(6); x.add(7);
                x.add(8);

                // connect nodes
                x.add(0, 5); x.add(9, 2); x.add(3,7);
                x.add(6,2); x.add(5,2);

                x.doSort();

                sort.add(1); sort.add(3); sort.add(6);
                sort.add(4); sort.add(5); sort.add(2);
                sort.add(9);
        }
} // end of Class ReferenceSortTest
```

Output


Workspace - Java - TopologicalSort/src/graph/ReferenceSortTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Test Field Assist Menu X2 Run Window H
Package Explorer
JUnit
Problems
Javadoc
Declaration
Console

Finished after 0.02 seconds

Runs: 3/3 Errors: 1 Failures: 1

graph.ReferenceSortTest [Runner: JUnit 4] (0.
graphErrorTest (0.000 s)
refCountTest (0.001 s)
incorrectTest (0.001 s)

```
<terminated> ReferenceSortTest [JUnit] C:\Program Files\Jav
refCountTest: [0, 1, 3, 4, 6, 8, 9]
```

Luke McCann: U1364096

Week12/13

Concurrent Systems

1) Will the test always terminate? I.e. is it certain that no matter how often you were to run the test it would always end in a finite length of time?

After extensive testing of the counter class, I can determine that the test will always terminate, eventually. Some operations take significantly longer than others.

For example:

If you are using a Counter(5,10) operating against a Counter(5,0)) the amount of time taken is less predictable and it can take significantly longer to terminate execution, although execution has always terminated at some point.

The time-scale can vary from a very short output trace to complete to a very long trace in the output console. This occurs due to the fact one counter is working against the other. Because of this they end up in a “tug of war” style of execution, in which both counters are moving in the opposite direction at the same rate until one gets an advantage over the other, in which then one completes and the other may count to its destination value.

For the logbook example (Counter(0,10) working against Counter(10,0)) the time is much more predictable and always ends in the same amount of finite time. In this test, Counter(0,10) and Counter(10,0) execute within the same finite time period every time they run.

This is because while Counter(0,10) begins at 0 the Counter(10,0) ends at 0 and vice versa. This results in less of a “tug of war” type of execution and one of the counters makes it to the value which ends their execution within the first two or three steps (as seen in the output below).

This means that the remaining counter is then left alone allowing it to either increment or decrement (depending which counter object is left) directly to the destination value, meaning no matter how many times you run the test, it will always execute within the same amount of time, and with the same amount of “steps” (14).

Final Answer: for the logbook example it will always end in the same finite length of time.

2) What is the shortest possible output for the test, in terms of the number of lines output?

The shortest possible output for the test Counter(0,10) Counter(10,0) appears to be set at 14 lines of output. After many times running this application I am confident to say that this is a constant number for the input when using this specific test. This is due to the same reason as the explanation above, due to the second counter that executes beginning at the terminating value for the first counter, counter N instantly reaches its destination value, leaving the counter X which has not yet finished to count directly to its destination number, without interruption. This means that the time taken to execute (and the lines of output) are a finite number which can be easily predicted and remains constant no matter how many times you execute the test as the same number of steps are taken.

Final Answer: 14

3) What is the largest possible value that the count can reach when the test is run?

Speaking overall, the largest possible value when you run the test is MAX (max being whichever value you have set as the highest within the counter).

In this case the number would be 10 as we have set our counters to count (0,10) and (10,0) in other words, the highest value the counters will ever reach is 10 (if the counter managed to reach higher than this they would fail to terminate as one counter would keep counting and never reach the end value as it somehow already surpassed it, implying there is a problem in the code).

Much like this, in the example Counter(5,0) Counter(5,10) the maximum value is still 10 (as this is the highest number specified), we could increase this value by creating a new Counter for instance Counter(0,20) Counter(20,0) would have a MAX value of 20 and Counter(0,15) Counter(15,0) would have a MAX value of 15.

If instead we use alternate values for each counter (for example we have Counter(5,20) and Counter(0,10) we see another way of viewing this. In this case the maximum possible value in which the Counter can reach is 20, as although the other counter specifies a maximum of 10 they are both counting to alternative MAX values, meaning that once the second counter reaches 10 it will terminate, however the first counter must reach its own MAX value of 20 in order to terminate and will continue counting until it has reached the value of 20, in which case it will then terminate.

Due to this we can deduce that, specific to the example we are currently working with, the maximum possible value is 10, however it would be safer (and more accurate) to say that the maximum possible value produced by Class Counter is MAX (the maximum specified value), on the contrary if we create two counters the maximum possible value produced is the greatest MAX value of the two objects:

I.e. with Counter(0,10) and Counter(0,15) the maximum value output would be 15.

Final Answer For Logbook Specific Test:

Maximum Output Value: 10

4) What is the lowest possible value that the count can reach when the test is run?

Much like the previous question, the lowest possible value operates in the same fashion as the largest possible value. In this specific test we can say that the lowest possible value is 0 however (again) it is more accurate to say that the minimum possible value is equal to MIN where MIN is equal to the lowest specified value (for instance Counter(5,20) has a minimum value of 5) however if there are two counters operating. Staying true to the previous question if we have two counters:

Counter(5,20) and Counter(20,0)

The lowest possible value for counter1 is 5, however for counter2 it is 0, therefore the lowest possible value in which can be displayed when the test is run is 0 as this is the lowest MIN value for count to reach in order to terminate the second counters execution. Again if the test were somehow to surpass this value and we had no more counters specifying a lower value, we could declare that we have a bug in our code somewhere as the test will never be able to terminate due to it being lower than the specified MIN value, this should never happen, and in this code is not possible to happen (the same goes for the largest possible value).

To clarify, the minimum possible value for this specific example is 0, as the counter terminates once it hits 0, however the more accurate description for overall implementation would be that MIN is equal to the smallest value provided to the counter objects.

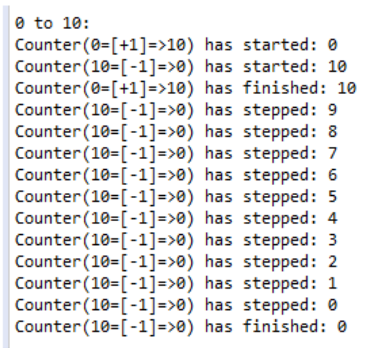
If there is more than one counter, the minimum possible output to be displayed is the smallest MIN value which has been provided between the two objects as one counter will always terminate/start at the lowest value it is given, this is the same for the other meaning that even if Counter(20,5) terminates at 5 and the Counter(10,20) starts at its lowest value 10, the lowest possible value to be output when running the test will be 5 as one counter will always terminate at 5 even if the other begins at 10, here we can say that Counter1 has a MIN value of 5 and Counter2 has a MIN value of 10, however the minimum output value will be 5 (the smallest MIN value between the two).

Final Answer For Logbook Specific Test:

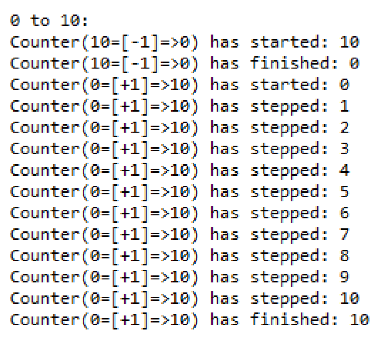
Minimum Output Value: 0

Outputs

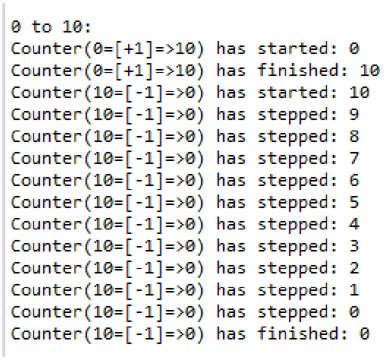
Test1: Counter(0 , 10) Counter(10 , 0)



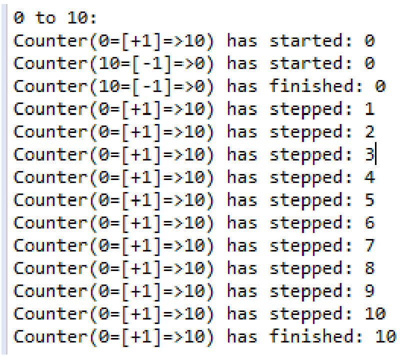
```
0 to 10:
Counter(0=[+1]=>10) has started: 0
Counter(10=[-1]=>0) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```



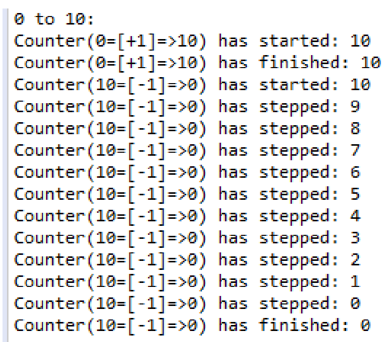
```
0 to 10:
Counter(10=[-1]=>0) has started: 10
Counter(10=[-1]=>0) has finished: 0
Counter(0=[+1]=>10) has started: 0
Counter(0=[+1]=>10) has stepped: 1
Counter(0=[+1]=>10) has stepped: 2
Counter(0=[+1]=>10) has stepped: 3
Counter(0=[+1]=>10) has stepped: 4
Counter(0=[+1]=>10) has stepped: 5
Counter(0=[+1]=>10) has stepped: 6
Counter(0=[+1]=>10) has stepped: 7
Counter(0=[+1]=>10) has stepped: 8
Counter(0=[+1]=>10) has stepped: 9
Counter(0=[+1]=>10) has stepped: 10
Counter(0=[+1]=>10) has finished: 10
```



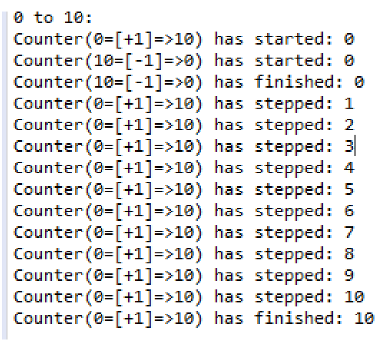
```
0 to 10:
Counter(0=[+1]=>10) has started: 0
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has started: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```



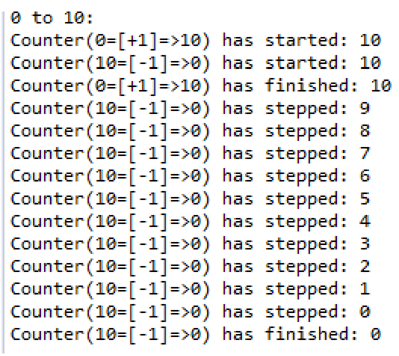
```
0 to 10:
Counter(0=[+1]=>10) has started: 0
Counter(10=[-1]=>0) has started: 0
Counter(10=[-1]=>0) has finished: 0
Counter(0=[+1]=>10) has stepped: 1
Counter(0=[+1]=>10) has stepped: 2
Counter(0=[+1]=>10) has stepped: 3
Counter(0=[+1]=>10) has stepped: 4
Counter(0=[+1]=>10) has stepped: 5
Counter(0=[+1]=>10) has stepped: 6
Counter(0=[+1]=>10) has stepped: 7
Counter(0=[+1]=>10) has stepped: 8
Counter(0=[+1]=>10) has stepped: 9
Counter(0=[+1]=>10) has stepped: 10
Counter(0=[+1]=>10) has finished: 10
```



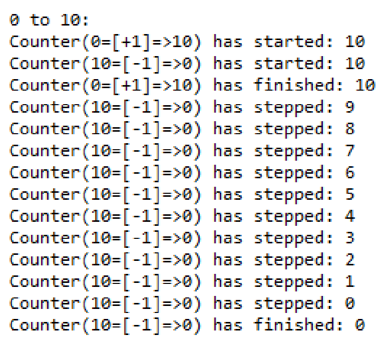
```
0 to 10:
Counter(0=[+1]=>10) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has started: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```



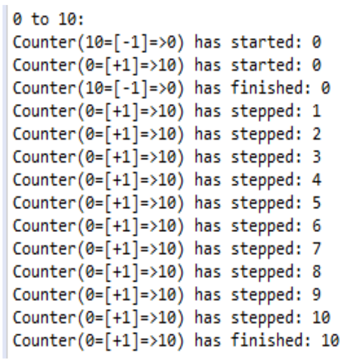
```
0 to 10:
Counter(0=[+1]=>10) has started: 0
Counter(10=[-1]=>0) has started: 0
Counter(10=[-1]=>0) has finished: 0
Counter(0=[+1]=>10) has stepped: 1
Counter(0=[+1]=>10) has stepped: 2
Counter(0=[+1]=>10) has stepped: 3
Counter(0=[+1]=>10) has stepped: 4
Counter(0=[+1]=>10) has stepped: 5
Counter(0=[+1]=>10) has stepped: 6
Counter(0=[+1]=>10) has stepped: 7
Counter(0=[+1]=>10) has stepped: 8
Counter(0=[+1]=>10) has stepped: 9
Counter(0=[+1]=>10) has stepped: 10
Counter(0=[+1]=>10) has finished: 10
```



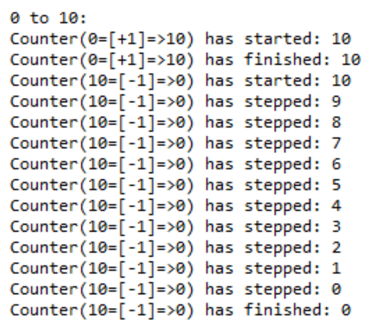
```
0 to 10:
Counter(0=[+1]=>10) has started: 10
Counter(10=[-1]=>0) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```



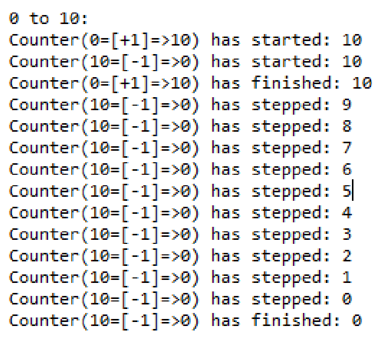
```
0 to 10:
Counter(0=[+1]=>10) has started: 10
Counter(10=[-1]=>0) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```



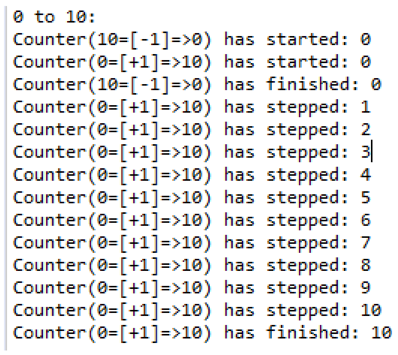
```
0 to 10:
Counter(10=[-1]=>0) has started: 0
Counter(0=[+1]=>10) has started: 0
Counter(10=[-1]=>0) has finished: 0
Counter(0=[+1]=>10) has stepped: 1
Counter(0=[+1]=>10) has stepped: 2
Counter(0=[+1]=>10) has stepped: 3
Counter(0=[+1]=>10) has stepped: 4
Counter(0=[+1]=>10) has stepped: 5
Counter(0=[+1]=>10) has stepped: 6
Counter(0=[+1]=>10) has stepped: 7
Counter(0=[+1]=>10) has stepped: 8
Counter(0=[+1]=>10) has stepped: 9
Counter(0=[+1]=>10) has stepped: 10
Counter(0=[+1]=>10) has finished: 10
```



```
0 to 10:
Counter(0=[+1]=>10) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has started: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```

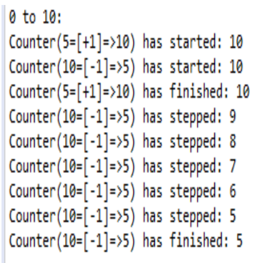


```
0 to 10:
Counter(0=[+1]=>10) has started: 10
Counter(10=[-1]=>0) has started: 10
Counter(0=[+1]=>10) has finished: 10
Counter(10=[-1]=>0) has stepped: 9
Counter(10=[-1]=>0) has stepped: 8
Counter(10=[-1]=>0) has stepped: 7
Counter(10=[-1]=>0) has stepped: 6
Counter(10=[-1]=>0) has stepped: 5
Counter(10=[-1]=>0) has stepped: 4
Counter(10=[-1]=>0) has stepped: 3
Counter(10=[-1]=>0) has stepped: 2
Counter(10=[-1]=>0) has stepped: 1
Counter(10=[-1]=>0) has stepped: 0
Counter(10=[-1]=>0) has finished: 0
```

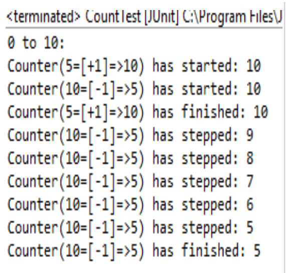


```
0 to 10:
Counter(10=[-1]=>0) has started: 0
Counter(0=[+1]=>10) has started: 0
Counter(10=[-1]=>0) has finished: 0
Counter(0=[+1]=>10) has stepped: 1
Counter(0=[+1]=>10) has stepped: 2
Counter(0=[+1]=>10) has stepped: 3
Counter(0=[+1]=>10) has stepped: 4
Counter(0=[+1]=>10) has stepped: 5
Counter(0=[+1]=>10) has stepped: 6
Counter(0=[+1]=>10) has stepped: 7
Counter(0=[+1]=>10) has stepped: 8
Counter(0=[+1]=>10) has stepped: 9
Counter(0=[+1]=>10) has stepped: 10
Counter(0=[+1]=>10) has finished: 10
```

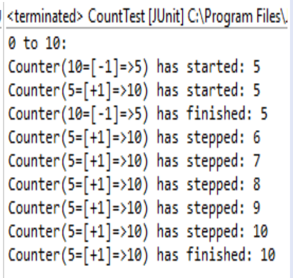
Test2: (5 , 10) Counter(10 , 5)



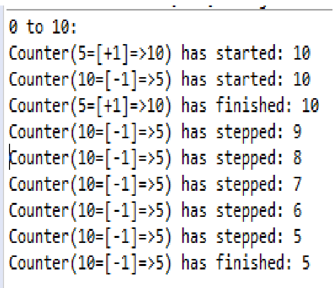
```
0 to 10:
Counter(5=[+1]=>10) has started: 10
Counter(10=[-1]=>5) has started: 10
Counter(5=[+1]=>10) has finished: 10
Counter(10=[-1]=>5) has stepped: 9
Counter(10=[-1]=>5) has stepped: 8
Counter(10=[-1]=>5) has stepped: 7
Counter(10=[-1]=>5) has stepped: 6
Counter(10=[-1]=>5) has stepped: 5
Counter(10=[-1]=>5) has finished: 5
```



```
<terminated> CountTest [JUnit] C:\Program Files\J
0 to 10:
Counter(5=[+1]=>10) has started: 10
Counter(10=[-1]=>5) has started: 10
Counter(5=[+1]=>10) has finished: 10
Counter(10=[-1]=>5) has stepped: 9
Counter(10=[-1]=>5) has stepped: 8
Counter(10=[-1]=>5) has stepped: 7
Counter(10=[-1]=>5) has stepped: 6
Counter(10=[-1]=>5) has stepped: 5
Counter(10=[-1]=>5) has finished: 5
```

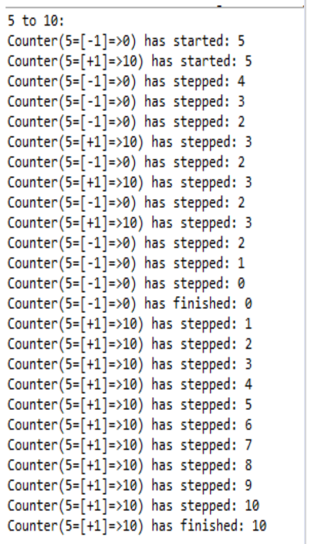


```
<terminated> CountTest [JUnit] C:\Program Files\
0 to 10:
Counter(10=[-1]=>5) has started: 5
Counter(5=[+1]=>10) has started: 5
Counter(10=[-1]=>5) has finished: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[+1]=>10) has stepped: 7
Counter(5=[+1]=>10) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[+1]=>10) has stepped: 10
Counter(5=[+1]=>10) has finished: 10
```

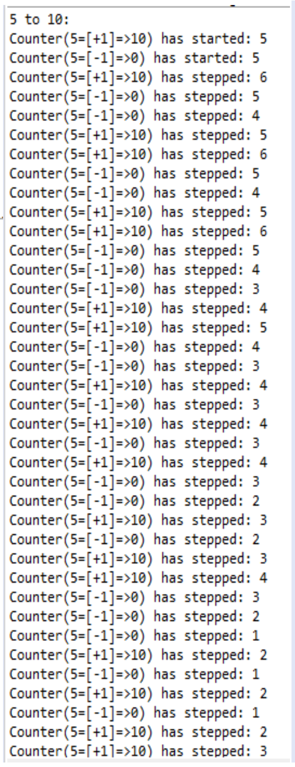


```
0 to 10:
Counter(5=[+1]=>10) has started: 10
Counter(10=[-1]=>5) has started: 10
Counter(5=[+1]=>10) has finished: 10
Counter(10=[-1]=>5) has stepped: 9
Counter(10=[-1]=>5) has stepped: 8
Counter(10=[-1]=>5) has stepped: 7
Counter(10=[-1]=>5) has stepped: 6
Counter(10=[-1]=>5) has stepped: 5
Counter(10=[-1]=>5) has finished: 5
```

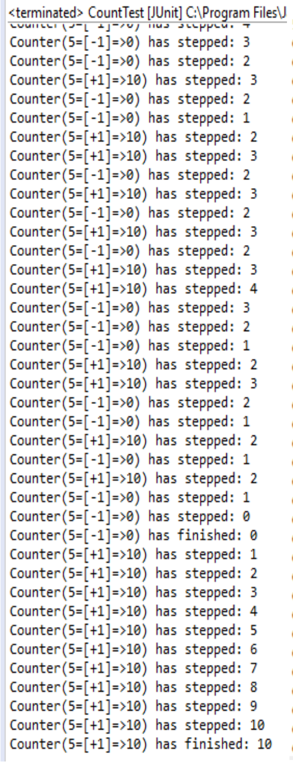
Test3: Outputs Counter(0 , 5) Counter (5 , 0)



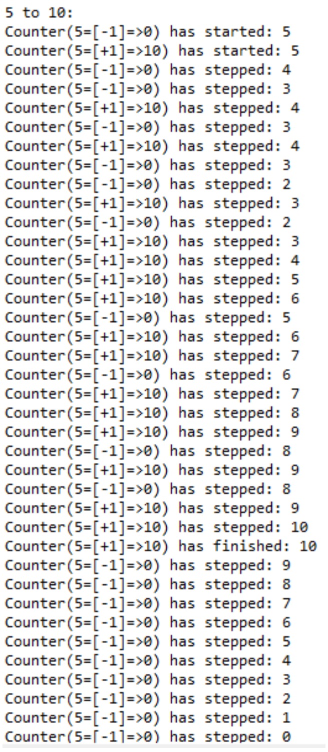
```
5 to 10:
Counter(5=[-1]=>0) has started: 5
Counter(5=[+1]=>10) has started: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[-1]=>0) has stepped: 0
Counter(5=[-1]=>0) has finished: 0
Counter(5=[+1]=>10) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[+1]=>10) has stepped: 7
Counter(5=[+1]=>10) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[+1]=>10) has stepped: 10
Counter(5=[+1]=>10) has finished: 10
```



```
5 to 10:
Counter(5=[+1]=>10) has started: 5
Counter(5=[-1]=>0) has started: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[-1]=>0) has stepped: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[-1]=>0) has stepped: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[-1]=>0) has stepped: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
```



```
<terminated> CountTest [JUnit] C:\Program Files\J
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[-1]=>0) has stepped: 0
Counter(5=[-1]=>0) has finished: 0
Counter(5=[+1]=>10) has stepped: 1
Counter(5=[+1]=>10) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[+1]=>10) has stepped: 7
Counter(5=[+1]=>10) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[+1]=>10) has stepped: 10
Counter(5=[+1]=>10) has finished: 10
```



```
5 to 10:
Counter(5=[-1]=>0) has started: 5
Counter(5=[+1]=>10) has started: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[+1]=>10) has stepped: 3
Counter(5=[+1]=>10) has stepped: 4
Counter(5=[+1]=>10) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[-1]=>0) has stepped: 5
Counter(5=[+1]=>10) has stepped: 6
Counter(5=[+1]=>10) has stepped: 7
Counter(5=[-1]=>0) has stepped: 6
Counter(5=[+1]=>10) has stepped: 7
Counter(5=[+1]=>10) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[-1]=>0) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[-1]=>0) has stepped: 8
Counter(5=[+1]=>10) has stepped: 9
Counter(5=[+1]=>10) has stepped: 10
Counter(5=[+1]=>10) has finished: 10
Counter(5=[-1]=>0) has stepped: 9
Counter(5=[-1]=>0) has stepped: 8
Counter(5=[-1]=>0) has stepped: 7
Counter(5=[-1]=>0) has stepped: 6
Counter(5=[-1]=>0) has stepped: 5
Counter(5=[-1]=>0) has stepped: 4
Counter(5=[-1]=>0) has stepped: 3
Counter(5=[-1]=>0) has stepped: 2
Counter(5=[-1]=>0) has stepped: 1
Counter(5=[-1]=>0) has stepped: 0
```

Week 14

Dekker's Algorithm

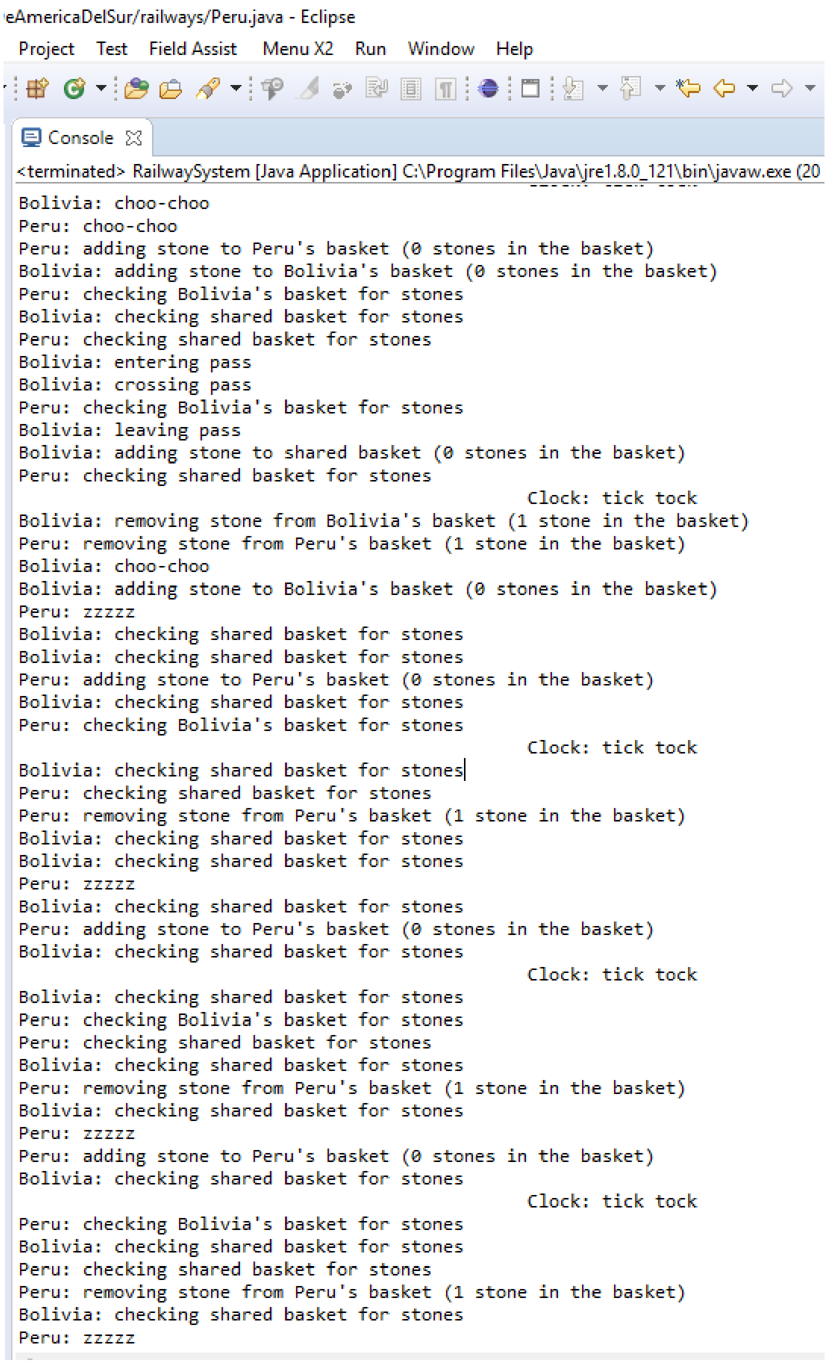
Bolivia.java

```java
package railways;

import errors.RailwaySystemError;
import errors.SetUpError;
import tools.Clock;
import tools.Delay;

public class Bolivia extends Railway {
	/**
	 * @throws RailwaySystemError if there is an error in constructing the delay
	 * Change the parameters of the Delay constructor in the call of the superconstructor
to
	 * change the behaviour of this railway.
	 */
	public Bolivia() throws SetUpError{
		super("Bolivia",new Delay(0.1,0.3));
	}

    /**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more than one
train in the
     * pass at any one time.
     */
    public void runTrain() throws RailwaySystemError {
    	Clock clock = getRailwaySystem().getClock();
    	Railway nextRailway = getRailwaySystem().getNextRailway(this);
    	while (!clock.timeOut()) {
    		choochoo(); // non critical section
    		getBasket().putStone(this); // put our stone in private basket

    		while(getSharedBasket().hasStone(this)) { // if other railways basket has stone

    			if(!getSharedBasket().hasStone(this)) { // and others turn
    			getBasket().takeStone(this); // remove out stone

    			    siesta(); // wait
    			    getBasket().putStone(this); // place our stone again
    			}
    		}
    		crossPass(); // critical section
    		getSharedBasket().putStone(this); // set process 1 turn
    		getBasket().takeStone(this); // remove our stone from private basket
    	}
    }
}
```

Peru.java

```
package railways;

import errors.RailwaySystemError;
import errors.SetUpError;
import tools.Clock;
import tools.Delay;

public class Peru extends Railway {
        /**
         * @throws RailwaySystemError if there is an error in constructing the delay
         * Change the parameters of the Delay constructor in the call of the superconstructor
to
         * change the behaviour of this railway.
         */
        public Peru() throws SetUpError{
                super("Peru",new Delay(0.1,0.3));
        }

    /**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more than one
train in the
     * pass at any one time.
     */
    public void runTrain() throws RailwaySystemError {
        Clock clock = getRailwaySystem().getClock();
        Railway nextRailway = getRailwaySystem().getNextRailway(this);

        while (!clock.timeOut()) {
                choochoo(); // non critical section
                getBasket().putStone(this); // put our stone in private basket

                while(nextRailway.getBasket().hasStone(this)) { // if other railways basket has
stone

                        if(getSharedBasket().hasStone(this)) {// and others turn
                            getBasket().takeStone(this); // remove our stone

                            siesta(); // wait
                            getBasket().putStone(this); // place our stone again
                        }
            }
            crossPass();  // critical section
            getSharedBasket().takeStone(this); // set process 2 turn
            getBasket().takeStone(this); // take our stone from private basket
        }
    }
}
```

Output


eAmericaDelSur/railways/Peru.java - Eclipse
Project Test Field Assist Menu X2 Run Window Help
Console

```
<terminated> RailwaySystem [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (20
Bolivia: choo-choo
Peru: choo-choo
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: checking shared basket for stones
Peru: checking shared basket for stones
Bolivia: entering pass
Bolivia: crossing pass
Peru: checking Bolivia's basket for stones
Bolivia: leaving pass
Bolivia: adding stone to shared basket (0 stones in the basket)
Peru: checking shared basket for stones
                                                Clock: tick tock
Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: choo-choo
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Peru: zzzzz
Bolivia: checking shared basket for stones
Bolivia: checking shared basket for stones
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: checking shared basket for stones
Peru: checking Bolivia's basket for stones
                                                Clock: tick tock
Bolivia: checking shared basket for stones
Peru: checking shared basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking shared basket for stones
Bolivia: checking shared basket for stones
Peru: zzzzz
Bolivia: checking shared basket for stones
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: checking shared basket for stones
                                                Clock: tick tock
Bolivia: checking shared basket for stones
Peru: checking Bolivia's basket for stones
Peru: checking shared basket for stones
Bolivia: checking shared basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking shared basket for stones
Peru: zzzzz
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: checking shared basket for stones
                                                Clock: tick tock
Peru: checking Bolivia's basket for stones
Bolivia: checking shared basket for stones
Peru: checking shared basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking shared basket for stones
Peru: zzzzz
```

Week 15

Semaphores

1) In the lecture it was said that in the implementation of bounded buffers using semaphores is the order of the critSec.P() and noElts.P(), in process consumer, was essential, but that the order of critSec.V() noElts.V(), in process producer, was not. Identify the corresponding piece of code in the Buffer class provided and make the change. Can you produce an error situation?

In order to produce a Deadlock Error, I switched around the criticalSection.poll() and noOfElements.poll() in order to create a situation in which the process attempts to check the buffer is free before receiving the number of elements, this creates a situation where the processes are constantly waiting.

2) Why does the error situation arise when the code is changed as described in question 1? Why does it not arise in the original code?

With the change in the code the process stops and awaits the data to be buffered. However, the process does not know if there is at least one buffer available as we are checking if the buffer is available before receiving information on the number of data items in the buffer.
This means that the process is constantly waiting for an available buffer as it cannot "see" if any of the other processes are currently using the buffer. The reason this did not arise before the change is that the process was receiving the number of items in the buffer first and therefore knew when it no longer needed to wait and was able to execute as normal.

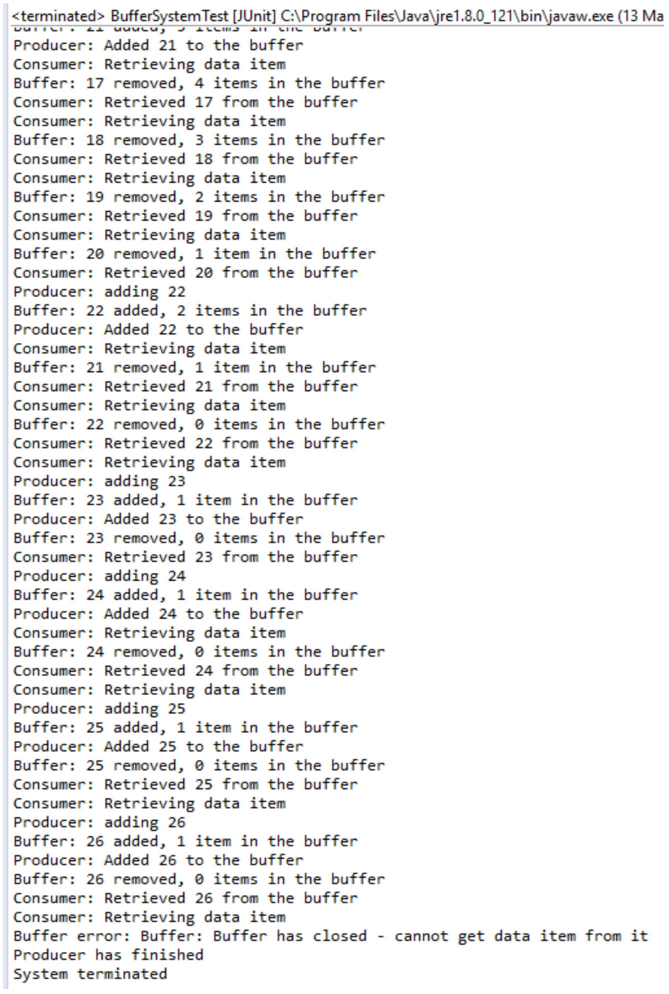
3) Is the order of the calls of poll in the Buffer class's put method also essential?

The order of the calls of "**poll()**" in the "**put**" method are also essential. This is because you cannot "**put**" to the buffer without first knowing if there is space for the datum to be placed in I.e. you cannot "**put**" to a buffer that you do not know has "free" space.

Before Editing

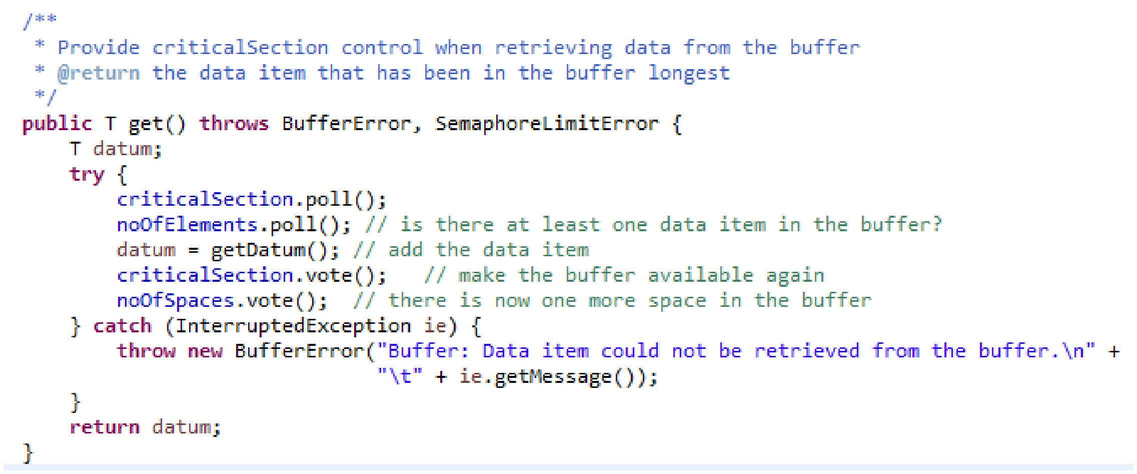
```
/**
 * Provide criticalSection control when retrieving data from the buffer
 * @return the data item that has been in the buffer longest
 */
public T get() throws BufferError, SemaphoreLimitError {
    T datum;
    try {
        noOfElements.poll(); // is there at least one data item in the buffer?
        criticalSection.poll();   // is the buffer available?
        datum = getDatum(); // add the data item
        criticalSection.vote();   // make the buffer available again
        noOfSpaces.vote();  // there is now one more space in the buffer
    } catch (InterruptedException ie) {
        throw new BufferError("Buffer: Data item could not be retrieved from the buffer.\n" +
                              "\t" + ie.getMessage());
    }
    return datum;
}
```

Output



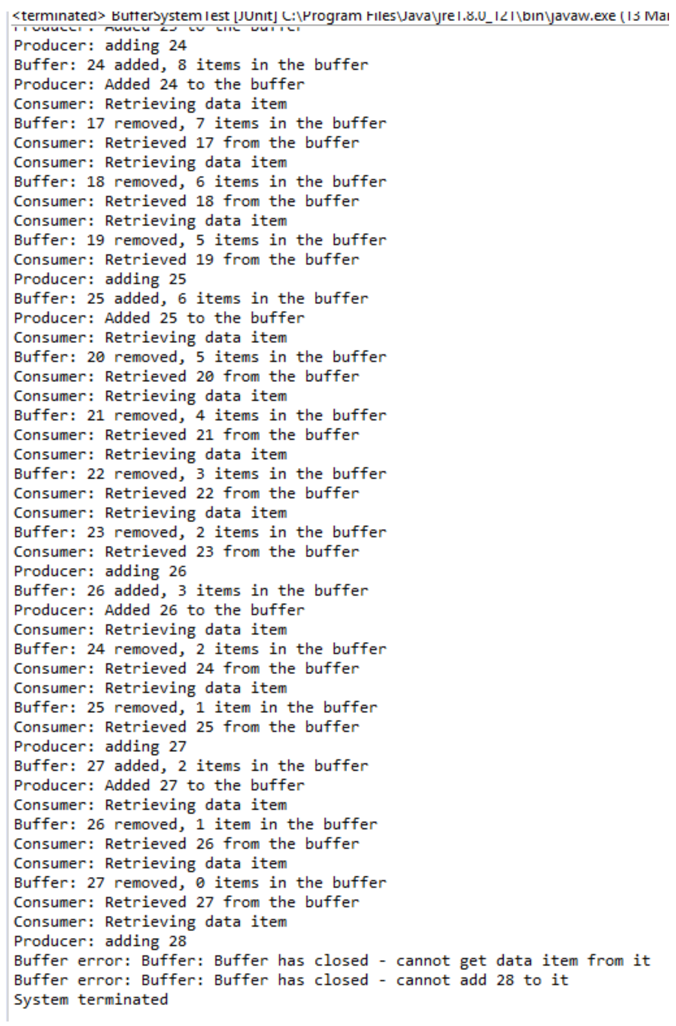
```
<terminated> BufferSystemTest [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (13 Ma
Producer: Added 21 to the buffer
Consumer: Retrieving data item
Buffer: 17 removed, 4 items in the buffer
Consumer: Retrieved 17 from the buffer
Consumer: Retrieving data item
Buffer: 18 removed, 3 items in the buffer
Consumer: Retrieved 18 from the buffer
Consumer: Retrieving data item
Buffer: 19 removed, 2 items in the buffer
Consumer: Retrieved 19 from the buffer
Consumer: Retrieving data item
Buffer: 20 removed, 1 item in the buffer
Consumer: Retrieved 20 from the buffer
Producer: adding 22
Buffer: 22 added, 2 items in the buffer
Producer: Added 22 to the buffer
Consumer: Retrieving data item
Buffer: 21 removed, 1 item in the buffer
Consumer: Retrieved 21 from the buffer
Consumer: Retrieving data item
Buffer: 22 removed, 0 items in the buffer
Consumer: Retrieved 22 from the buffer
Consumer: Retrieving data item
Producer: adding 23
Buffer: 23 added, 1 item in the buffer
Producer: Added 23 to the buffer
Buffer: 23 removed, 0 items in the buffer
Consumer: Retrieved 23 from the buffer
Producer: adding 24
Buffer: 24 added, 1 item in the buffer
Producer: Added 24 to the buffer
Consumer: Retrieving data item
Buffer: 24 removed, 0 items in the buffer
Consumer: Retrieved 24 from the buffer
Consumer: Retrieving data item
Producer: adding 25
Buffer: 25 added, 1 item in the buffer
Producer: Added 25 to the buffer
Buffer: 25 removed, 0 items in the buffer
Consumer: Retrieved 25 from the buffer
Consumer: Retrieving data item
Producer: adding 26
Buffer: 26 added, 1 item in the buffer
Producer: Added 26 to the buffer
Buffer: 26 removed, 0 items in the buffer
Consumer: Retrieved 26 from the buffer
Consumer: Retrieving data item
Buffer error: Buffer: Buffer has closed - cannot get data item from it
Producer has finished
System terminated
```

After Editing



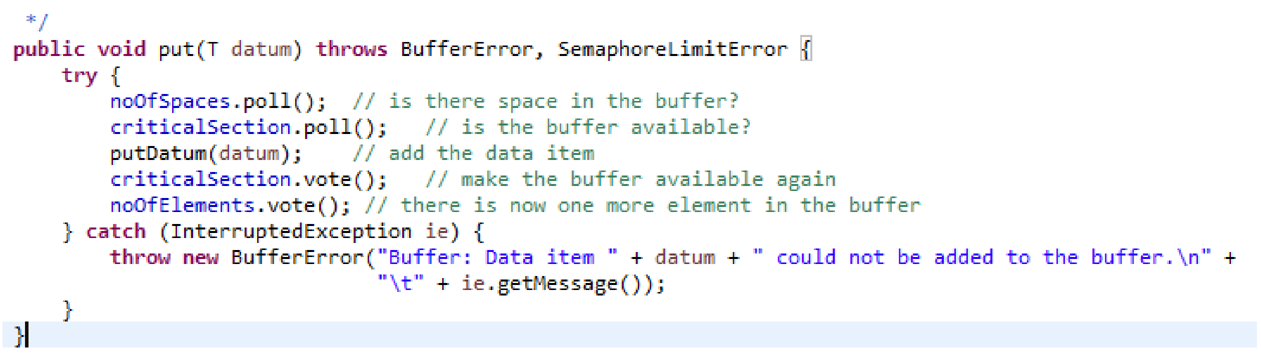
```java
/**
 * Provide criticalSection control when retrieving data from the buffer
 * @return the data item that has been in the buffer longest
 */
public T get() throws BufferError, SemaphoreLimitError {
    T datum;
    try {
        criticalSection.poll();
        noOfElements.poll(); // is there at least one data item in the buffer?
        datum = getDatum(); // add the data item
        criticalSection.vote();   // make the buffer available again
        noOfSpaces.vote();  // there is now one more space in the buffer
    } catch (InterruptedException ie) {
        throw new BufferError("Buffer: Data item could not be retrieved from the buffer.\n" +
                              "\t" + ie.getMessage());
    }
    return datum;
}
```

Output



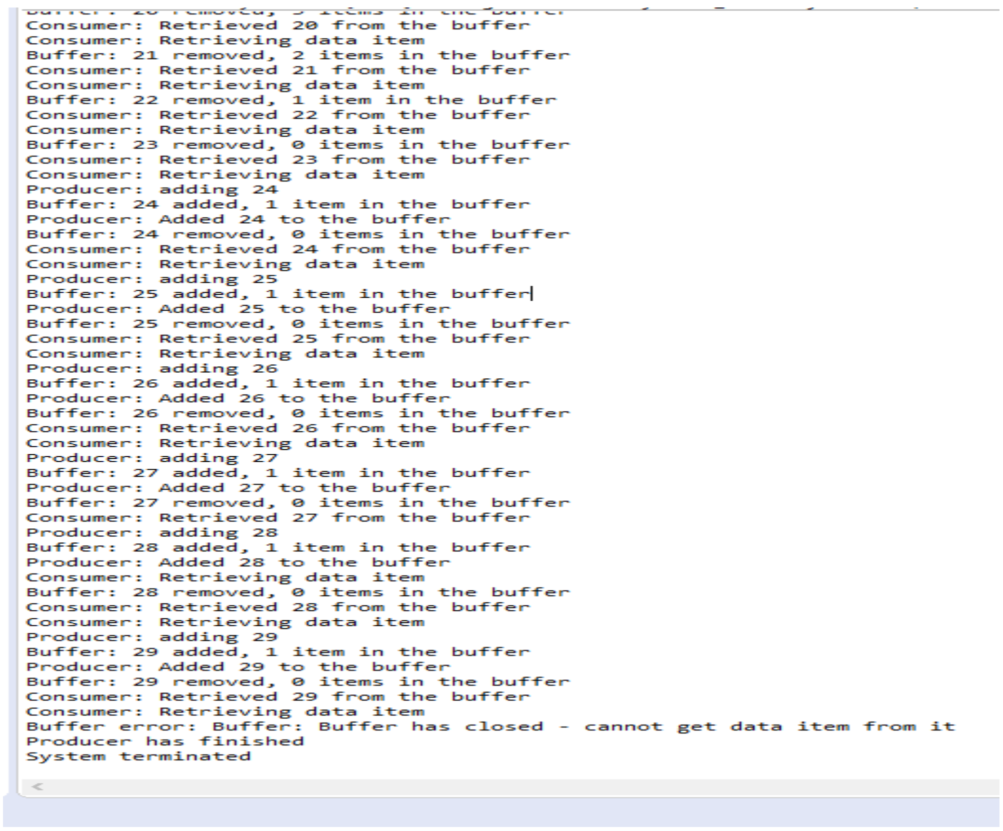
```
<terminated> BufferSystemTest [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (13 Mar
Producer: adding 24
Buffer: 24 added, 8 items in the buffer
Producer: Added 24 to the buffer
Consumer: Retrieving data item
Buffer: 17 removed, 7 items in the buffer
Consumer: Retrieved 17 from the buffer
Consumer: Retrieving data item
Buffer: 18 removed, 6 items in the buffer
Consumer: Retrieved 18 from the buffer
Consumer: Retrieving data item
Buffer: 19 removed, 5 items in the buffer
Consumer: Retrieved 19 from the buffer
Producer: adding 25
Buffer: 25 added, 6 items in the buffer
Producer: Added 25 to the buffer
Consumer: Retrieving data item
Buffer: 20 removed, 5 items in the buffer
Consumer: Retrieved 20 from the buffer
Consumer: Retrieving data item
Buffer: 21 removed, 4 items in the buffer
Consumer: Retrieved 21 from the buffer
Consumer: Retrieving data item
Buffer: 22 removed, 3 items in the buffer
Consumer: Retrieved 22 from the buffer
Consumer: Retrieving data item
Buffer: 23 removed, 2 items in the buffer
Consumer: Retrieved 23 from the buffer
Producer: adding 26
Buffer: 26 added, 3 items in the buffer
Producer: Added 26 to the buffer
Consumer: Retrieving data item
Buffer: 24 removed, 2 items in the buffer
Consumer: Retrieved 24 from the buffer
Consumer: Retrieving data item
Buffer: 25 removed, 1 item in the buffer
Consumer: Retrieved 25 from the buffer
Producer: adding 27
Buffer: 27 added, 2 items in the buffer
Producer: Added 27 to the buffer
Consumer: Retrieving data item
Buffer: 26 removed, 1 item in the buffer
Consumer: Retrieved 26 from the buffer
Consumer: Retrieving data item
Buffer: 27 removed, 0 items in the buffer
Consumer: Retrieved 27 from the buffer
Consumer: Retrieving data item
Producer: adding 28
Buffer error: Buffer: Buffer has closed - cannot get data item from it
Buffer error: Buffer: Buffer has closed - cannot add 28 to it
System terminated
```

Before Editing Put



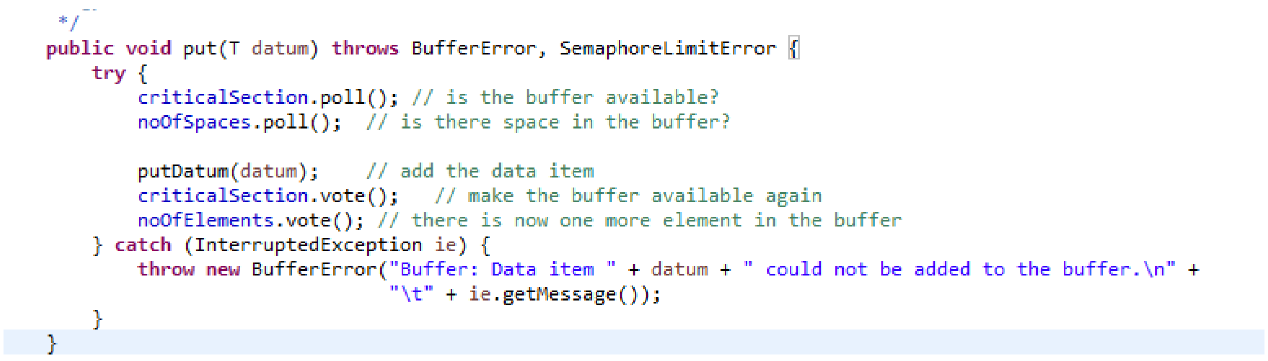
```
 */
public void put(T datum) throws BufferError, SemaphoreLimitError {
    try {
        noOfSpaces.poll();  // is there space in the buffer?
        criticalSection.poll();   // is the buffer available?
        putDatum(datum);    // add the data item
        criticalSection.vote();   // make the buffer available again
        noOfElements.vote(); // there is now one more element in the buffer
    } catch (InterruptedException ie) {
        throw new BufferError("Buffer: Data item " + datum + " could not be added to the buffer.\n" +
                        "\t" + ie.getMessage());
    }
}
```

Output



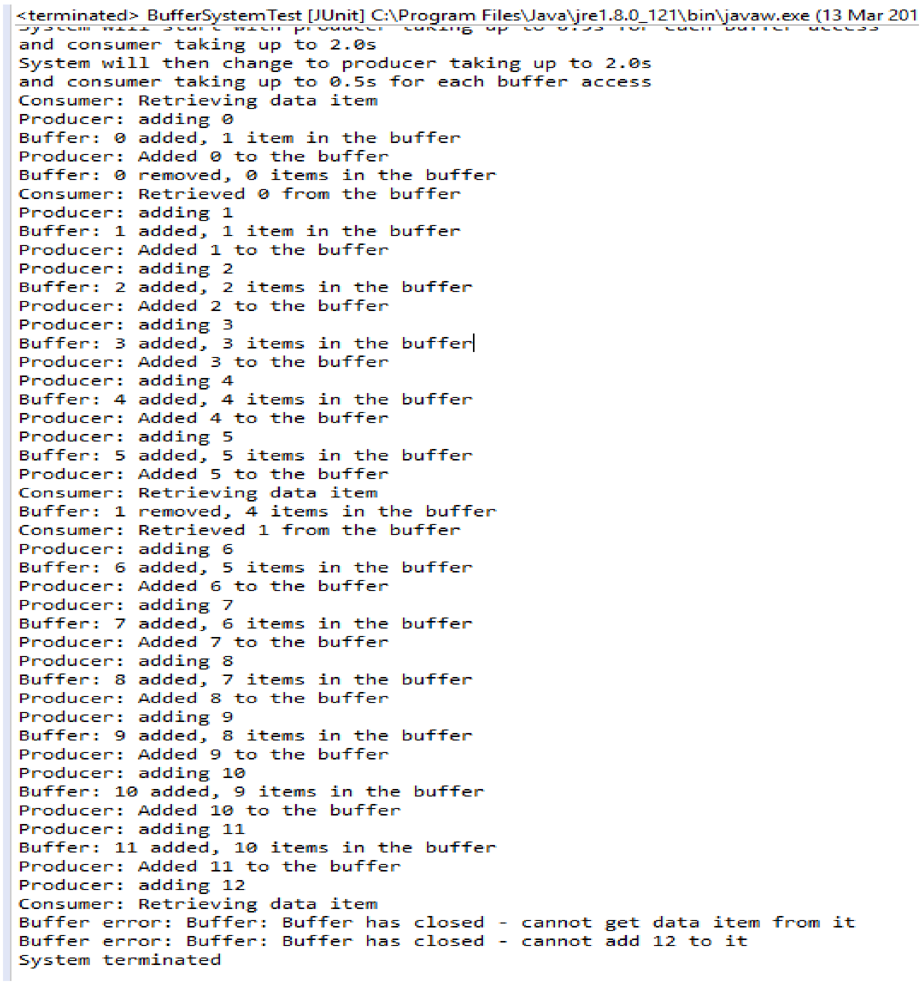
```
Consumer: Retrieved 20 from the buffer
Consumer: Retrieving data item
Buffer: 21 removed, 2 items in the buffer
Consumer: Retrieved 21 from the buffer
Consumer: Retrieving data item
Buffer: 22 removed, 1 item in the buffer
Consumer: Retrieved 22 from the buffer
Consumer: Retrieving data item
Buffer: 23 removed, 0 items in the buffer
Consumer: Retrieved 23 from the buffer
Consumer: Retrieving data item
Producer: adding 24
Buffer: 24 added, 1 item in the buffer
Producer: Added 24 to the buffer
Buffer: 24 removed, 0 items in the buffer
Consumer: Retrieved 24 from the buffer
Consumer: Retrieving data item
Producer: adding 25
Buffer: 25 added, 1 item in the buffer
Producer: Added 25 to the buffer
Buffer: 25 removed, 0 items in the buffer
Consumer: Retrieved 25 from the buffer
Consumer: Retrieving data item
Producer: adding 26
Buffer: 26 added, 1 item in the buffer
Producer: Added 26 to the buffer
Buffer: 26 removed, 0 items in the buffer
Consumer: Retrieved 26 from the buffer
Consumer: Retrieving data item
Producer: adding 27
Buffer: 27 added, 1 item in the buffer
Producer: Added 27 to the buffer
Buffer: 27 removed, 0 items in the buffer
Consumer: Retrieved 27 from the buffer
Producer: adding 28
Buffer: 28 added, 1 item in the buffer
Producer: Added 28 to the buffer
Consumer: Retrieving data item
Buffer: 28 removed, 0 items in the buffer
Consumer: Retrieved 28 from the buffer
Consumer: Retrieving data item
Producer: adding 29
Buffer: 29 added, 1 item in the buffer
Producer: Added 29 to the buffer
Buffer: 29 removed, 0 items in the buffer
Consumer: Retrieved 29 from the buffer
Consumer: Retrieving data item
Buffer error: Buffer: Buffer has closed - cannot get data item from it
Producer has finished
System terminated
```

After Editing Put



```
     */
    public void put(T datum) throws BufferError, SemaphoreLimitError {
        try {
            criticalSection.poll(); // is the buffer available?
            noOfSpaces.poll();  // is there space in the buffer?

            putDatum(datum);    // add the data item
            criticalSection.vote();   // make the buffer available again
            noOfElements.vote(); // there is now one more element in the buffer
        } catch (InterruptedException ie) {
            throw new BufferError("Buffer: Data item " + datum + " could not be added to the buffer.\n" +
                                  "\t" + ie.getMessage());
        }
    }
```

Output



```
<terminated> BufferSystemTest [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (13 Mar 201
System will start with producer taking up to 0.5s for each buffer access
and consumer taking up to 2.0s
System will then change to producer taking up to 2.0s
and consumer taking up to 0.5s for each buffer access
Consumer: Retrieving data item
Producer: adding 0
Buffer: 0 added, 1 item in the buffer
Producer: Added 0 to the buffer
Buffer: 0 removed, 0 items in the buffer
Consumer: Retrieved 0 from the buffer
Producer: adding 1
Buffer: 1 added, 1 item in the buffer
Producer: Added 1 to the buffer
Producer: adding 2
Buffer: 2 added, 2 items in the buffer
Producer: Added 2 to the buffer
Producer: adding 3
Buffer: 3 added, 3 items in the buffer
Producer: Added 3 to the buffer
Producer: adding 4
Buffer: 4 added, 4 items in the buffer
Producer: Added 4 to the buffer
Producer: adding 5
Buffer: 5 added, 5 items in the buffer
Producer: Added 5 to the buffer
Consumer: Retrieving data item
Buffer: 1 removed, 4 items in the buffer
Consumer: Retrieved 1 from the buffer
Producer: adding 6
Buffer: 6 added, 5 items in the buffer
Producer: Added 6 to the buffer
Producer: adding 7
Buffer: 7 added, 6 items in the buffer
Producer: Added 7 to the buffer
Producer: adding 8
Buffer: 8 added, 7 items in the buffer
Producer: Added 8 to the buffer
Producer: adding 9
Buffer: 9 added, 8 items in the buffer
Producer: Added 9 to the buffer
Producer: adding 10
Buffer: 10 added, 9 items in the buffer
Producer: Added 10 to the buffer
Producer: adding 11
Buffer: 11 added, 10 items in the buffer
Producer: Added 11 to the buffer
Producer: adding 12
Consumer: Retrieving data item
Buffer error: Buffer: Buffer has closed - cannot get data item from it
Buffer error: Buffer: Buffer has closed - cannot add 12 to it
System terminated
```

Week16

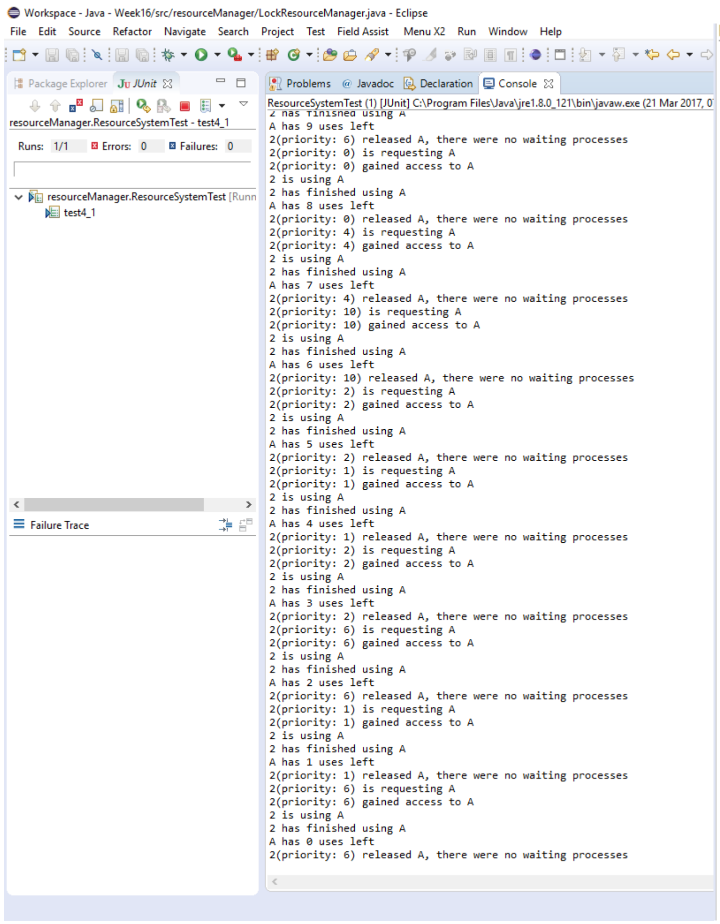
Monitors

LockResourceManager.java

```
package resourceManager;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class LockResourceManager extends BasicResourceManager {


        private Lock lock = new ReentrantLock();
        private Condition[] queues = new Condition[NO_OF_PRIORITIES];

        private boolean resourceInUse;

        public LockResourceManager(Resource resource, int maxUses) {
                super(resource, maxUses);
                for(int priority = 0; priority < NO_OF_PRIORITIES; priority++) {
                        queues[priority] = lock.newCondition();
                }
        }

     /**
      * Requests the use of the manager's resource, with the specified priority.
      * If the resource is in use the requesting user will  wait for the resource to be 
released.
      * otherwise useResource() is called and the resource is used
      * @param priority the priority level at which the resource is being requested.
      * @throws ResourceError if the implementing code throws an InterruptedException error.
      */
        @Override
        public void requestResource(int priority) throws ResourceError {
                lock.lock(); // lock critical section
                while(resourceInUse) { // if other resource is using
                        increaseNumberWaiting(priority);  // join queue
                        try {
                                queues[priority].await();  // obtain this monitor
                                lock.unlock();  // free the critical sections
                                queues[priority].wait();    // and sleep
                                lock.lock(); // lock critical section again
                        } catch(InterruptedException error) {
                                throw new ResourceError(getResourceName() +
                                                    " was interrupted while waiting in priorty " + 
priority + " queue - " + error.getMessage());
                        }
                }
                resourceInUse = true;  // lock the resource (mark it as in use)
                lock.unlock();  // free the critical sections
        }

        /**
         * Release this managers resource, if any users are waiting for the
         * resource a waiting user with the highest priority is woken
         * @throws ResourceError if the implementing code throws and InterruptedException
         */
        @Override
        public int releaseResource() throws ResourceError {
                lock.lock(); // lock critical section
                int highestWaitingPriority = NONE_WAITING; // start looking process asleep
                for(int i = 0; i > NO_OF_PRIORITIES; i++) { // check priorities
                        if(getNumberWaiting(i) > 0) { // if there are processes waiting
                                highestWaitingPriority = i; // set highestWaitingPriority
                        }
                }
```

```
                    if(highestWaitingPriority != NONE_WAITING) { // if found sleeping
process
                            decreaseNumberWaiting(highestWaitingPriority);  // it is about
to be woken
                            queues[highestWaitingPriority].notify(); // wake it
                    }
                    resourceInUse = false; // mark as not in use
                    lock.unlock();  // unlock critical section
                    return highestWaitingPriority;
            }


        /**
         * checks the number of users waiting are < 0
         * if the number of users waiting are > 0 returns true
         * else returns false
         * @return true if number waiting > 0
         */
        public boolean userWaiting() {
                if(this.getNumberWaiting(NO_OF_PRIORITIES) > 0) {
                        return true;
                }
                else {
                        return false;
                }
        }

}
```

Outputs


Workspace - Java - Week16/src/resourceManager/LockResourceManager.java - Eclipse

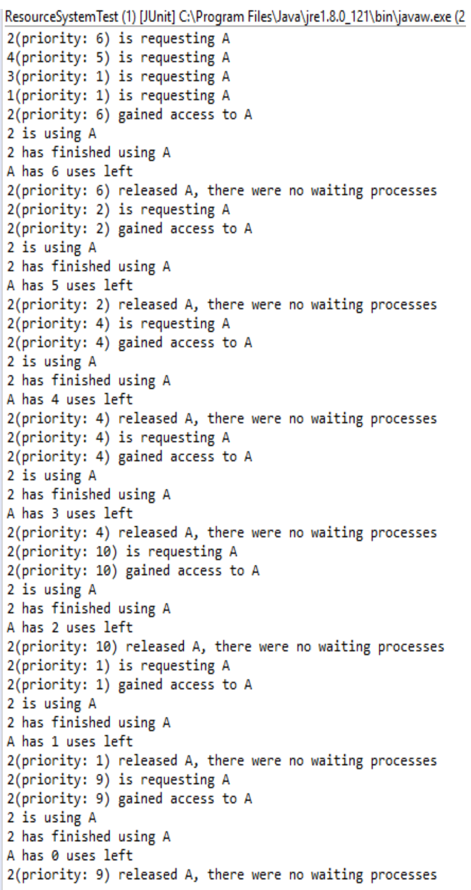
resourceManager.ResourceSystemTest - test4_1

Runs: 1/1 Errors: 0 Failures: 0

resourceManager.ResourceSystemTest [Runn
test4_1

ResourceSystemTest (1) [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (21 Mar 2017, 0

```
2 has finished using A
A has 9 uses left
2(priority: 6) released A, there were no waiting processes
2(priority: 0) is requesting A
2(priority: 0) gained access to A
2 is using A
2 has finished using A
A has 8 uses left
2(priority: 0) released A, there were no waiting processes
2(priority: 4) is requesting A
2(priority: 4) gained access to A
2 is using A
2 has finished using A
A has 7 uses left
2(priority: 4) released A, there were no waiting processes
2(priority: 10) is requesting A
2(priority: 10) gained access to A
2 is using A
2 has finished using A
A has 6 uses left
2(priority: 10) released A, there were no waiting processes
2(priority: 2) is requesting A
2(priority: 2) gained access to A
2 is using A
2 has finished using A
A has 5 uses left
2(priority: 2) released A, there were no waiting processes
2(priority: 1) is requesting A
2(priority: 1) gained access to A
2 is using A
2 has finished using A
A has 4 uses left
2(priority: 1) released A, there were no waiting processes
2(priority: 2) is requesting A
2(priority: 2) gained access to A
2 is using A
2 has finished using A
A has 3 uses left
2(priority: 2) released A, there were no waiting processes
2(priority: 6) is requesting A
2(priority: 6) gained access to A
2 is using A
2 has finished using A
A has 2 uses left
2(priority: 6) released A, there were no waiting processes
2(priority: 1) is requesting A
2(priority: 1) gained access to A
2 is using A
2 has finished using A
A has 1 uses left
2(priority: 1) released A, there were no waiting processes
2(priority: 6) is requesting A
2(priority: 6) gained access to A
2 is using A
2 has finished using A
A has 0 uses left
2(priority: 6) released A, there were no waiting processes
```


ResourceSystemTest (1) [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (2

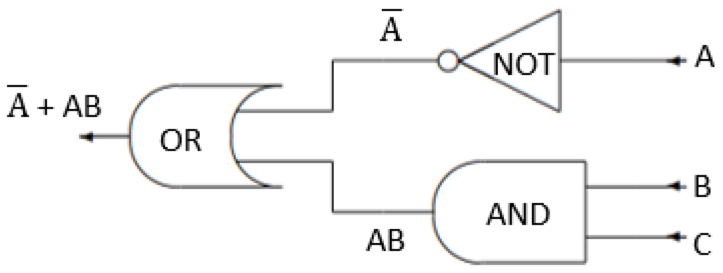
```
2(priority: 6) is requesting A
4(priority: 5) is requesting A
3(priority: 1) is requesting A
1(priority: 1) is requesting A
2(priority: 6) gained access to A
2 is using A
2 has finished using A
A has 6 uses left
2(priority: 6) released A, there were no waiting processes
2(priority: 2) is requesting A
2(priority: 2) gained access to A
2 is using A
2 has finished using A
A has 5 uses left
2(priority: 2) released A, there were no waiting processes
2(priority: 4) is requesting A
2(priority: 4) gained access to A
2 is using A
2 has finished using A
A has 4 uses left
2(priority: 4) released A, there were no waiting processes
2(priority: 4) is requesting A
2(priority: 4) gained access to A
2 is using A
2 has finished using A
A has 3 uses left
2(priority: 4) released A, there were no waiting processes
2(priority: 10) is requesting A
2(priority: 10) gained access to A
2 is using A
2 has finished using A
A has 2 uses left
2(priority: 10) released A, there were no waiting processes
2(priority: 1) is requesting A
2(priority: 1) gained access to A
2 is using A
2 has finished using A
A has 1 uses left
2(priority: 1) released A, there were no waiting processes
2(priority: 9) is requesting A
2(priority: 9) gained access to A
2 is using A
2 has finished using A
A has 0 uses left
2(priority: 9) released A, there were no waiting processes
```

Week17/18

Circuits

1) What is the matrix for the following circuit?

This Circuit has a matrix of: $\begin{bmatrix} 0\,0\,0\,0\,1\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix}$



$OR * (NOT \otimes AND)$ = $\begin{bmatrix} 0\,0\,0\,0\,1\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix}$

Calculations

NOT = $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ AND = $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ OR = $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

NOT $\otimes$ AND = $\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$

$OR * (NOT \otimes AND)$ = $\begin{bmatrix} 0\,0\,0\,0\,1\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix}$

Additional Learning Notes (Practice and Necessary information)

Bits

True and False

$$true \equiv |1\rangle \equiv \begin{bmatrix}0\\1\end{bmatrix} \qquad false \equiv |0\rangle \equiv \begin{bmatrix}1\\0\end{bmatrix}$$

Bit sequences are calculated as the Tensor Product of bit matrices

Note: a byte $|0110\ 1001\rangle$ has 256 rows as for each bit added the size of the Matrix doubles

Bit Sequences

If we have a sequence of $|01\rangle$;

$$|01\rangle \equiv |0\rangle \otimes |1\rangle \equiv \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\begin{bmatrix}0\\1\end{bmatrix}\\0\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

We take the Tensor Product of the 0 Matrix and the 1 Matrix.

Gates

Not

$$not = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

The application of the NOT gate is simple Matrix Multiplication;

$$not\ true \equiv not|1\rangle \equiv \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} * \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} \equiv |0\rangle \equiv false$$

$$not\ false \equiv not|0\rangle \equiv \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} * \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} \equiv |1\rangle \equiv true$$

And

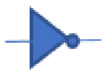
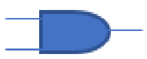
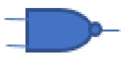
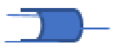
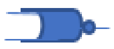
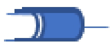
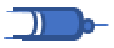
$$and = \begin{bmatrix}1 & 1 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

In this case, we times each item along the top of Matrix 1 with each one down matrix 2 adding them together, and then the same for the second row;

$$and\ |0,1\rangle \equiv \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \equiv |0\rangle$$

Truth Tables

Original

Gate	NOT	AND	NAND	OR	NOR	XOR	XNOR
Algebraic Expression	$\bar{X}$	XY	$\overline{XY}$	$X+Y$	$\overline{X+Y}$	$X \otimes Y$	$\overline{X \otimes Y}$
Symbol							
Truth Table	X | Z 0 | 1 1 | 0	X Y | Z 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	X Y | Z 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	X Y | Z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	X Y | Z 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	X Y | Z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	X Y | Z 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Matrices (Truth Tables)

NOT

X $\bar{X}$	0 1	1 0	INPUT
$\lvert\bar{X}\rangle$	$\lvert 1\rangle$	$\lvert 0\rangle$	OUTPUT MATRIX
	0 1	1 0	Expanded Matrix

AND

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
X ^ Y	*0*	*0*	*0*	*1*	*OUTPUT*
$\|X \wedge Y\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	*OUTPUT MATRIX*
	1	*1*	*1*	*0*	*EXPANDED MATRIX*
	0	*0*	*0*	*1*	

NAND

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
$\overline{XY}$	*1*	*1*	*1*	*0*	*OUTPUT*
$\overline{\|XY\rangle}$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	*OUTPUT MATRIX*
	0	*0*	*0*	*1*	*EXPANDED MATRIX*
	1	*1*	*1*	*0*	

OR

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
$X + Y$	*0*	*1*	*1*	*1*	*OUTPUT*
$\|X + Y\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	*OUTPUT MATRIX*
	1	*0*	*0*	*0*	*EXPANDED MATRIX*
	0	*1*	*1*	*1*	

NOR

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
$X \otimes Y$	*1*	*0*	*0*	*0*	*OUTPUT*
$\|X \otimes Y\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	*OUTPUT MATRIX*
	0	*1*	*1*	*1*	*EXPANDED MATRIX*
	1	*0*	*0*	*0*	

XOR

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
$X \otimes Y$	*0*	*1*	*1*	*0*	*OUTPUT*
$\lvert X \otimes Y\rangle$	$\lvert 0\rangle$	$\lvert 1\rangle$	$\lvert 1\rangle$	$\lvert 0\rangle$	*OUTPUT MATRIX*
	1	*0*	*0*	*1*	*EXPANDED MATRIX*
	0	*1*	*1*	*0*	

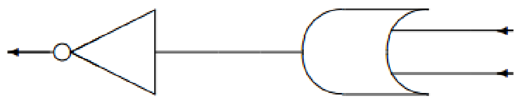
XNOR

X	*0*	*0*	*1*	*1*	*INPUT*
Y	*0*	*1*	*0*	*1*	
$X \otimes Y$	*1*	*0*	*0*	*1*	*OUTPUT*
$\lvert X \otimes Y\rangle$	$\lvert 1\rangle$	$\lvert 0\rangle$	$\lvert 0\rangle$	$\lvert 1\rangle$	*OUTPUT MATRIX*
	0	*1*	*1*	*0*	*EXPANDED MATRIX*
	1	*0*	*0*	*1*	

Sequential Circuits

Sequential Circuits use the matrix product;

NAND = not * and

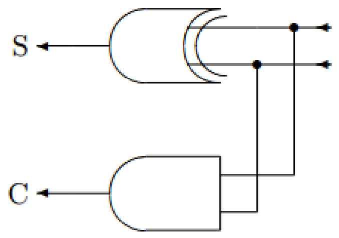


$$nand = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Parallel Circuits

Circuits in parallel use the Tensor Product;

Since it goes through both at the same time we need to take the XOR matrix and the AND matrix and take the Tensor Product of the two;



Complex Numbers

Complex Conjugate

To calculate the Complex Conjugate the sign is simply flipped for the "imaginary" part of the equation, for example;

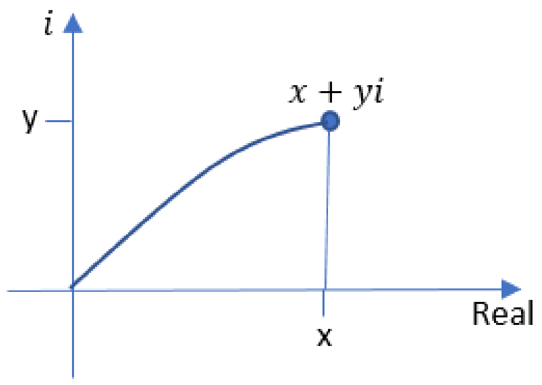
$$\overline{-5+3i} = -5-3i \qquad \overline{5-3i} = 5+3i$$

Magnitude

The Magnitude of a Complex Number is calculated by taking the "real" part of the Complex Number and the size of the "imaginary" part (ignoring the "i") we then square both components and take the square root of both numbers, for example;

$$|x+yi| = \sqrt{x^2 + y^2}$$

Another way to think of this would be as a representation on a two-directional plain;



The square of the hypotenuse is equal to the sum of the square on the other two sides, when drawing out a 2D plain representation of complex numbers, the Magnitude of the Complex Number is the distance from the origin (0) to the Complex Number.

Addition/Subtraction

When adding, and subtracting, we simply add the real and imaginary parts separately;

$$(5 + 2i) + (3 - i) = 8 - i \qquad (5 + 3i) - (2 - 3i) = 3 + 6i$$

Multiplication

When multiplying, we multiply out the brackets.

$$\begin{aligned}(3 + 2i)(2 - 4i) &= 3(2 - 4i) + 2i(2 - 4i)\\ &= 6 - 12i + 4i - 8(i\times i)\\ &= 6 - 8i - 8(-1)\\ &= 6 - 8i + 8\\ &= 14 - 8i\end{aligned}$$

Division

We start by multiplying both the numerator and denominator by the conjugate of the denominator

$$\begin{aligned}\frac{3+2i}{2-4i} &= \frac{3+2i}{2-4i}\times\frac{2+4i}{2+4i}\\ &= \frac{(3+2i)}{(2-4i)}\frac{(2+4i)}{(2+4i)}\\ &= \frac{6+16i+8i^2}{2^2+4^2}\\ &= \frac{6+16i-8}{4+16}\\ &= \frac{-2+16i}{20}\end{aligned}$$

Matrices

Addition/Subtraction

Matrix addition/subtraction is just "piecewise" calculation, you add/subtract the current number to the corresponding item, note the matrices MUST be the same size;

$$\begin{bmatrix}3 & -2 & 9\\ 10 & 2 & -9\\ 22 & -11 & 4\end{bmatrix} + \begin{bmatrix}4 & -1 & 6\\ -2 & -5 & 2\\ 10 & -2 & 12\end{bmatrix} = \begin{bmatrix}7 & -3 & 15\\ 8 & -3 & -7\\ 32 & -14 & 16\end{bmatrix}$$

Multiplication

There are various types of Multiplication for Matrices;

Scalar

The matrices is multiplied by a set number;

$$5 \cdot \begin{bmatrix} 2 & -4 & 7 \\ 9 & 2 & -3 \\ 12 & 0 & -4 \end{bmatrix} = \begin{bmatrix} 10 & -20 & 35 \\ 45 & 10 & -15 \\ 60 & 0 & -20 \end{bmatrix}$$

Matrix Product

We can also multiply matrices together:
Here we have two Matrices one of size 5 by 3 the other of size 3 by 4, in order for multiplication to be possible the number in the middle of the two must be the same (in this case 3), the result in this case is a 5 by 4 Matrices;

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ -2 & 6 & 5 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} * \begin{bmatrix} \cdot & \cdot & 3 & \cdot \\ \cdot & \cdot & 7 & \cdot \\ \cdot & \cdot & 1 & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 41 & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

If we wanted to get the 3rd entry of the 2nd row, we take the 2nd row of the first matrix and the 3rd column of the 2nd matrix, we then do pairwise multiplication and add the product together i.e;

$$-2\times3 + 6\times7 + 5\times1 = 41$$

Tensor Product

The tensor product has no restrictions on the sizes of the Matrices. In order to take the Tensor Product of two matrices we take the first matrices, and for each of the entries in this first matrix create a copy of the second matrix. Lastly, we take the scalar product of the second matrix.

$$\begin{bmatrix} 5 & 0 \\ -2 & -4 \\ 0 & 1 \end{bmatrix} \otimes [3 \quad -4] = \begin{bmatrix} 5 \cdot [3 \quad -4] & 0 \cdot [3 \quad -4] \\ -2 \cdot [3 \quad -4] & -4 \cdot [3 \quad -4] \\ 0 \cdot [3 \quad -4] & 1 \cdot [3 \quad -4] \end{bmatrix} = \begin{bmatrix} 15 & -20 & 0 & 0 \\ -6 & 8 & -12 & 16 \\ 0 & 0 & 3 & -4 \end{bmatrix}$$

Unary Operators

Transpose

The Transpose operator “flips” the rows and the columns;

$$\begin{bmatrix} 5 & 4 & 2 \\ 4 & 2 & 9 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 5 & 4 \\ 4 & 2 \\ 2 & 9 \end{bmatrix}$$

Conjugate

If we have Matrices which contain Complex Numbers, it is possible to take the piecewise Conjugate of the Matrix, to do this we take the Conjugate of each element within that Matrix;

$$\begin{bmatrix} 1+i & -8+2i & 1 \\ -2+1i & 9 & 0 \end{bmatrix} = \begin{bmatrix} 1-i & -8+2i & 1 \\ -2+1i & 9 & 0 \end{bmatrix}$$

Adjoint

The Adjoint is the Transpose of the Conjugate, or the Conjugate of the Transpose

Week 19/20

Hadamard Gates

The Hadamard Gate creates “Quantum Superposition’s” in the sense if you apply the Hadamard gate to a Qubit of value 100% |0⟩ you get a 50/50 superposition of a 0 and 1 in phase, when applied to |1⟩ you get a 50/50 out of phase superposition of 0 and 1, when applied twice in succession final state is always equal to the initial state.

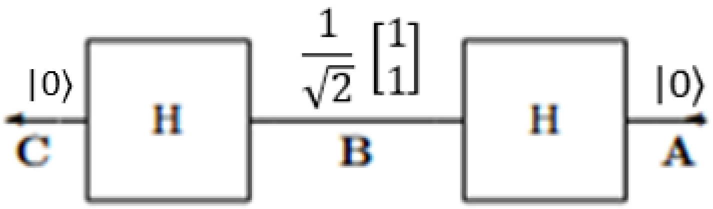
1) If |ψ⟩ is a pure state (i.e. |ψ⟩ = |0⟩ or |ψ⟩ = |1⟩, what happens if |ψ⟩ is passed as input to a circuit consisting of two Hadamard gates in sequence.

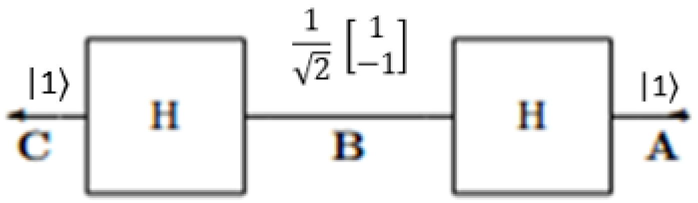
If |ψ⟩ is passed as input into a circuit consisting of two Hadamard gates the process is as follows;

The first Hadamard Gate puts the pure state |ψ⟩ into a superposition, in this position it has a probability of being 0, 1 or both at the same time, once this superposition of |ψ⟩ has passed through the second Hadamard Gate it returns to its initial pure state.

As an example, the diagram below shows two inputs into a circuit consisting of two Hadamard Gates in succession. As you can see we whichever pure state is input is first processed into a superposition, however once the second Hadamard Gate process this superposition, it will always return to its initial state.

Although these states can be expressed in “Traditional Notation” Ket Notation allows us to keep track of both |1⟩ whether or not |ψ⟩ belongs to singular or dual position, not only this but Ket notation is much clearer for performing calculations as the vector representations allow for much simpler calculations to be performed within the equation.





Transformation Tables

We can see the transformation as:

Qubit States:	0	1
Point A	100 % (0)	100% (1)
Point B	50% (0) 50% (1)	50% (1) 50% (0)
Point C	100% (0)	100% (1)

However it is more accurate to show as (and simpler to calculate with);

Qubit States:	0	1
Point A	[1,0]	[0,1]
Point B	[1/sqrt(2)] * [1,-1;1,1]	[1/sqrt(2)] * [1,-1;1,1]
Point C	[1,0]	[0,1]

Calculations

Traditional Notation

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{2}(|0\rangle + |1\rangle) + \frac{1}{2}(|0\rangle - |1\rangle) = |0\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

$$H\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{2}(|0\rangle + |1\rangle) - \frac{1}{2}(|0\rangle - |1\rangle) = |1\rangle$$

Bra-ket Notation

$$\text{Hadamard Gate} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

Position B =

INPUT: Position A: $|0\rangle$ =

$$\text{Hadamard Gate} * |0\rangle = \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\times1 + & 1\times0\\1\times1 + & (-1)\times0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

INPUT: Position A: $|1\rangle$ =

$$\text{Hadamard Gate} * |1\rangle = \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\times0 + & 1\times1\\1\times0 + & (-1)\times1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

Position C =

INPUT: Position A: $|0\rangle$

$$\text{Hadamard Gate} * \text{B} = \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\times\frac{1}{\sqrt{2}}\begin{bmatrix}1\times1 + & 1\times1\\1\times1 + (-1)\times1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

INPUT: Position A: $|1\rangle$

$$\text{Hadamard Gate} * \text{B} = \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} * \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{\sqrt{2}}\times\frac{1}{\sqrt{2}}\begin{bmatrix}1\times1 + & 1\times(-1)\\1\times1 + & (-1)\times(-1)\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

Revisited Items

Below are the weeks which I have worked on some more in order to improve my skills and expand my knowledge, these are mostly the areas at which I have struggled with and decided to come back to for a second try, or to improve on.

Dekker's Algorithm (Working Revisit)

Bolivia.java

```
package railways;

import errors.RailwaySystemError;
import errors.SetUpError;
import tools.Clock;
import tools.Delay;

public class Bolivia extends Railway {
        /**
         * @throws RailwaySystemError if there is an error in constructing the delay
         * Change the parameters of the Delay constructor in the call of the superconstructor
to
         * change the behaviour of this railway.
         */
        public Bolivia() throws SetUpError{
                super("Bolivia",new Delay(0.1,0.3));
        }

    /**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more than one
train in the
     * pass at any one time.
     */
    public void runTrain() throws RailwaySystemError {
        Clock clock = getRailwaySystem().getClock();
        Railway nextRailway = getRailwaySystem().getNextRailway(this);

        while (!clock.timeOut()) {
                choochoo(); // non critical section
                getBasket().putStone(this); // put our stone in private basket

                while(nextRailway.getBasket().hasStone(this)) { // if other railways basket has
stone

                        if(!getSharedBasket().hasStone(this)) { // and others turn
                        getBasket().takeStone(this); // remove our stone

                            siesta(); // wait
                            getBasket().putStone(this); // place our stone again
                        }
                }
                crossPass(); // critical section
                getSharedBasket().putStone(this); // set process 1 turn
                getBasket().takeStone(this); // remove our stone from private basket
        }
```

Peru.java

```
package railways;

import errors.RailwaySystemError;
import errors.SetUpError;
import tools.Clock;
import tools.Delay;

public class Peru extends Railway {
	/**
	 * @throws RailwaySystemError if there is an error in constructing the delay
	 * Change the parameters of the Delay constructor in the call of the superconstructor 
to
	 * change the behaviour of this railway.
	 */
	public Peru() throws SetUpError{
		super("Peru",new Delay(0.1,0.3));
	}

    /**
     * Run the train on the railway.
     * This method provides (incorrect) synchronisation attempting to avoid more than one 
train in the
     * pass at any one time.
     */
    public void runTrain() throws RailwaySystemError {
    	Clock clock = getRailwaySystem().getClock();
    	Railway nextRailway = getRailwaySystem().getNextRailway(this);

    	while (!clock.timeOut()) {
    		choochoo(); // non critical section
    		getBasket().putStone(this); // put our stone in private basket

    		while(nextRailway.getBasket().hasStone(this)) { // if other railways basket has 
stone

    			if(getSharedBasket().hasStone(this)) {// and others turn
    			    getBasket().takeStone(this); // remove our stone

    			    siesta(); // wait
    			    getBasket().putStone(this); // place our stone again
    			}
    		}
    		crossPass();  // critical section
    		getSharedBasket().takeStone(this); // set process 2 turn
    		getBasket().takeStone(this); // take our stone from private basket
    	}
    }
```

Output

```
                                       Clock: tick tock
Peru: choo-choo
Bolivia: choo-choo
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Peru: checking shared basket for stones
Bolivia: checking shared basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
                                       Clock: tick tock
Peru: checking shared basket for stones
Bolivia: zzzzz
Peru: checking Bolivia's basket for stones
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Peru: entering pass
Peru: crossing pass
Bolivia: checking Peru's basket for stones
Peru: leaving pass
Peru: adding stone to shared basket (0 stones in the basket)
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking shared basket for stones
                                       Clock: tick tock
Peru: choo-choo
Bolivia: checking Peru's basket for stones
Peru: adding stone to Peru's basket (0 stones in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: checking shared basket for stones
Bolivia: checking Peru's basket for stones
Peru: checking shared basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: checking shared basket for stones
Peru: zzzzz
                                       Clock: tick tock
Bolivia: checking Peru's basket for stones
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: entering pass
Bolivia: crossing pass
Peru: checking Bolivia's basket for stones
Peru: checking shared basket for stones
Bolivia: leaving pass
Bolivia: removing stone from shared basket (1 stone in the basket)
Peru: removing stone from Peru's basket (1 stone in the basket)
Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
Peru: zzzzz
Bolivia: choo-choo
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
                                       Clock: tick tock
Peru: adding stone to Peru's basket (0 stones in the basket)
Bolivia: checking Peru's basket for stones
Peru: checking Bolivia's basket for stones
Bolivia: checking shared basket for stones
Peru: checking shared basket for stones
Bolivia: removing stone from Bolivia's basket (1 stone in the basket)
Peru: checking Bolivia's basket for stones
Bolivia: zzzzz
```

```
                                       Clock: tick tock
Peru: entering pass
Peru: crossing pass
Bolivia: adding stone to Bolivia's basket (0 stones in the basket)
Peru: leaving pass
Peru: adding stone to shared basket (0 stones in the basket)
Bolivia: checking Peru's basket for stones
Peru: removing stone from Peru's basket (1 stone in the basket)
Peru: choo-choo
```

DepthFirstTraversal (Working Revisit)

DepthFirstTraversal.java

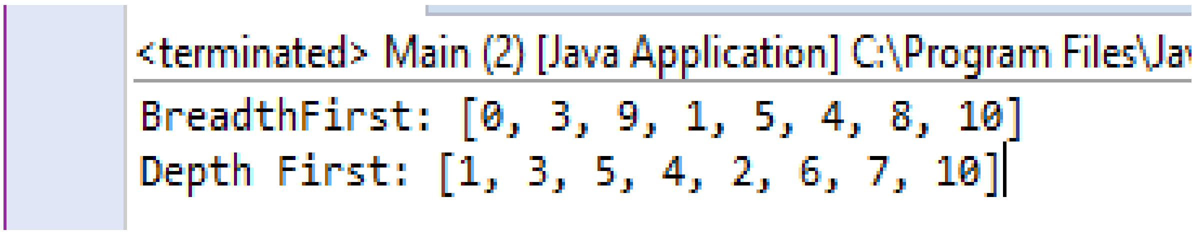
```java
package graph;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Queue;

public class DepthFirstTraversal<T> extends AdjacencyGraph<T> implements Traversal<T> {

        private Queue<T> toDo;
        private List<T> traversal;

        public ArrayList<T> traverse() throws GraphError {
                traversal = new ArrayList<T>();
                toDo = new ArrayDeque<T>();
                T node = getUnvisitedNode();
                while(node != null) {
                        visitNode(node);
                        traverseToDoList();
                        node = getUnvisitedNode();
                }
                return (ArrayList<T>) traversal;
        }

        /**
         * Check if a node has been visited  A node counts as visited if it is:
         * <ul>
         *   <li> either in the traversal or has been visited
         *   <li> is in the to do list, and scheduled to be visited
         * </ul>
         * @return this node has been visited or is scheduled to be visited
         */
        private boolean isVisited(T node) {
                return
                                traversal.contains(node)
                                || toDo.contains(node);
        }

        /**
         * Get the next "unvisited" node.  This method will actually also count nodes
         * that are scheduled to be visited (i.e. in the to do list) as visited.
         * @return a node that has not yet been visited, and that is not yet scheduled to be
visited, or return null if no such node exists
         */
        private T getUnvisitedNode() {
                for (T node: getNodes()) {
                        if (!isVisited(node)) {
                                return node;
                        }
                }
                // checked all nodes, and there are no unvisited nodes
                return null; // so return null
        }

        /**
         * Use the to do list to traverse the graph.
         * @throws GraphError if node is not a node in the graph
         */
        private void traverseToDoList() throws GraphError {
                while (toDo.size() > 0) {
                        T node = toDo.remove();
                        visitNode(node);
                }
        }

        /**
         * Visit a node.
         * @throws GraphError if node is not a node in the graph
         */
        private void visitNode(T node) throws GraphError {
                if (isVisited(node)) return;
                traversal.add(node);
                for (T neighbour: neighbours(node)) {
                        if (!isVisited(neighbour))
                                toDo.add(neighbour);
                }
        }
}
```

Main.java

```
package graph;

import java.util.List;

public class Main {

        public static void main(String[] args) {
                 BredthTest();
                 DepthTest();
        }

        public static void DepthTest() {
                 try {
                        DepthFirstTraversal<Integer> dt = new DepthFirstTraversal<Integer>();

                 dt.add(1);
                 dt.add(3);
                 dt.add(2);
                 dt.add(4);
                 dt.add(7);
                 dt.add(5);
                 dt.add(6);
                 dt.add(10);

                 dt.add(1,5);
                 dt.add(3,4);
                 dt.add(6,1);
                 dt.add(10,4);
                 dt.add(1,3);


                 List<Integer> dOutput = dt.traverse();
                 System.out.println("Depth First: " + dOutput);
                  } catch (GraphError e) {
                          e.printStackTrace();
                  }
        }

        public static void BredthTest() {
                try {
                        BreadthFirstTraversal<Integer> bt = new
BreadthFirstTraversal<Integer>();
                        bt.add(0);
                        bt.add(1);
                        bt.add(3);
                        bt.add(4);
                        bt.add(5);
                 bt.add(8);
                 bt.add(9);
                 bt.add(10);

                 bt.add(1,5);
                 bt.add(3,9);
                 bt.add(1,1);
                 bt.add(10,5);
                 bt.add(0,3);
```

```
				List<Integer> bOutput = bt.traverse();


				System.out.println("BreadthFirst: " + bOutput);

			} catch(GraphError e) {
				e.printStackTrace();
			}
		}
}
```

Output



```
<terminated> Main (2) [Java Application] C:\Program Files\Ja
BreadthFirst: [0, 3, 9, 1, 5, 4, 8, 10]
Depth First: [1, 3, 5, 4, 2, 6, 7, 10]
```

Additional Materials

Below are the additional items which, although relevant to the module, failed to fit into the theme of previous weeks;

Perlin Noise Algorithm (1 Dimensional)

Outside of University, I have worked on various projects. One of these projects entails an android game, consisting of procedurally generated terrain. During my research for this project I came across the “Perlin Noise Algorithm”, this is an algorithm created by Ken Perlin (Early 1980’s), however the implementation I have worked on is the extended version of this algorithm, published in 2002, this implementation of the algorithm is to generate random 1dimensional terrain for an Android Game, working using the extended version of the Perlin Noise Algorithm. This algorithm allows for natural phenomena in procedural generation (such as caves, rivers etc) along with fire and water effects, it can be used in up to 4Dimensions, meaning it can be used to animate, creating realistic terrain effects in games programming, due to working with Unity the following code is written in C#.

```
using UnityEngine;
using System.Collections;

public class PerlinNoise {

	long seed;

	public PerlinNoise(long seed)
	{
		this.seed = seed;
	}

    /**
     * random - returns a randomly generated seed of a certain range.
     * @param int x - minimum to int start at.
     * @param int range - maximum possible value.
     */
	private int random(long x, int range)
	{
		return (int)(((x+seed)^5) % range);
```

```
        }

    /**
     * getNoise - Returns a random int between the chunkSize and Range
     * @param int x - minimum int to start at.
     * @param int range - the maximum possible value.
     */
        public int getNoise(int x, int range)
    {
                int chunkSize = 16;
                float noise = 0;

                range /= 2;

                while(chunkSize > 0){
                        int chunkIndex = x / chunkSize;
                        float prog = (x % chunkSize) / (chunkSize * 1f);
                        float lRandom = random(chunkIndex, range);
                        float rRandom = random(chunkIndex + 1, range);


                        noise += (1-prog) * lRandom + prog * rRandom;

                        chunkSize /= 2;
                        range /= 2;

                        range = Mathf.Max(1,range);
                }

                return (int)Mathf.Round(noise);
        }
    }
```

Genetic Algorithms

Traditional Genetic Algorithms use a Darwinian approach to solving a problem in which the answer is unknown. These Algorithms are inspired by evolutionary sciences and take a form of natural selection as an implementation to solve a problem. Today it is common knowledge that in order for evolution to take place, three key factors must be in action;

. Variation
. Heredity
. Selection

Based on these key factors Genetic Algorithms aim to solve problems in which an answer is not known by the programmer. This is an important goal in the field of Evolutionary Computing as it not only allows us to solve problems "unsolvable" by humans, but also to come up with new, innovative and unique concepts and ideas that a human may not normally consider.

Research currently developing in robotics and movement uses Genetic Algorithms to allow robots to calculate the most efficient mode of movement for itself., sometimes coming up with surprising results.
Other implementations of genetic algorithms are seen in systems optimisation and have key uses in many forms of AI including prediction algorithms and organisational approaches in combination with neural networks.

Genetic Algorithms begin with an initial population; this population must have some form of variation in order for the solution to "Evolve", in other words the populations Chromosomes must be able to Mutate.

In this example I am using a basic generation of integers within an array to evolve a specific array I refer to as "Gene Code". if there is not enough variation within a Genetic Algorithm the algorithm will eventually "settle" at a constant in which it cannot possibly get any closer to the intended target. This is a problem which is solved through to use of a mutation, as we know from the rules of evolution – some variation is needed in order for evolution to be possible – hence, at least a minute amount of mutation is needed in order to add the much needed variety to our populations, allowing for a solution to “Evolve” for itself.

Variation is more complex than this first step however. The **Initialize()** method first of all creates an initial population of N with random genetic materials (this is our digital DNA) with this we can then add our **crossover()** method. The **crossover()** method allows us to take the most successful candidates (the “Elites”) from our digital gene pool, and randomly mix them, this creates a new type of variation, in which the most accurate (or dominant) “genes” are passed from both parent elements to the child.

In order for this to work however we have to give each Chromosome a fitness rating. A fitness rating allows us to rate each generation of elements on how well they perform the task they have been created to do or, in this case, how close of a match our Chromosome is to the “target” Chromosome.

After this step selection occurs, in this case I will select 2 parents (this can be done with more or even just one, however using one would just create a direct copy, removing the chances of anymore variation), here I will take the traditional approach by assigning a probability for each to be picked as a parent dependent on their fitness score, we can then “filter” out the “weaker” genes allowing for our Chromosome to move closer to the target, although some generations can possibly move backwards, the aim is not necessarily to find our answer as fast as possible, we could do that much quicker without a Genetic Algorithm for this example, but to allow the Algorithm to find a solution for itself (as mentioned earlier usually when using Genetic Algorithms we do not “know” the answer).

Below is the code for the first implementation of a Genetic Algorithm, in the future I plane to attempt a Generic solution to this problem, in which can be translated into other formats easily in order to evolve an answer to other problems, I have been recently experimenting with evolving pathways for AI game objects using vectors and hope to be able to use an implementation of a Genetic Algorithm to evolve a solution to the problem of which the terrain around the “objects” generates randomly, and each generation must evolve a better pathway in order to avoid the obstacles.

Chromosome.java

```
/**
 * Holds the main functionality of the GeneticAlgorithm
 * such as evoloution, mutation, reproduction
 * @author Luke McCann
 * @date December 2016
 *
 */
public class Algorithm {

        public static int[] targetGeneCode = {0,1,0,1,1,1,0,0,1,1,1,0,0,1};
        public static int population = 8;
        public static int noOfEliteGenes = 0;

        public static int maxSurvivors = 4;
        private static double mutationRate = 0.5;  // percentage expressed as double value

        public Algorithm() {
        }

        public Algorithm(int populationDensity, int[] targetGeneticCode) {
                Algorithm.population = populationDensity;
                Algorithm.targetGeneCode = targetGeneticCode;
        }

        /**
         * evolves the population
         * @param population - population to mutate
         * @return mutatedPopulation
         */
        public Population evolve(Population population) {
                return mutateAllPop(crossOver(population));
        }

        /**
         * Applies an element of randomness specified by the
         * mutationRate, needed in order for an answer to evolve
         * @param population - population to mutate
         * @return mutantChromosome - the <tt>Chromosome</tt> which has been mutated
         */
        private Chromosome mutateChromosome(Chromosome x) {
                Chromosome mutantChromosome =
                               new Chromosome(targetGeneCode.length);

                for(int i = 0; i < x.getGenes().length; i++) {
                    if(Math.random() < mutationRate) {
                            if(Math.random() < 0.5) {
                                    mutantChromosome.getGenes()[i] = 1;
                            }
                            else if(Math.random() > 0.5){
                                    mutantChromosome.getGenes()[i] = 0;
                            }
                            else {
                                    mutantChromosome.getGenes()[i] = x.getGenes()[i];
                            }
                    }
            }
            return mutantChromosome;
    }

    /**
     * creates a new population to mutate
     * @param population - population to mutate
     * @return mutatePop - mutated population
     */
    private Population mutateAllPop(Population population) {
            Population mutatePop =
                            new Population(population.getAllChromosomes().length);

            for(int i = 0; 0 < noOfEliteGenes; i++) {

                    mutatePop.getAllChromosomes()[i] =
                                    population.getAllChromosomes()[i]; //
            }
            for(int i = noOfEliteGenes; i <
                            population.getAllChromosomes().length; i++) {
                    mutatePop.getAllChromosomes()[i] =
mutateChromosome(population.getAllChromosomes()[i]);
            }
            return mutatePop;
    }


    /**
     * Randomly selects <tt>Chromosomes</tt>
     * to reuse: only those with the highest fitness
     * survive
     * @return competingPop - the population competing for highest fitness
     */
    private Population selectSurivingGenetics(Population population) {
            Population competingPop =
                            new Population(maxSurvivors);

            for(int i = 0; i < maxSurvivors ; i++) {
                    competingPop.getAllChromosomes()[i] =
                                    population.getAllChromosomes()

    [(int)(Math.random()*population.getAllChromosomes().length)];
            }
            competingPop.sortByFitness();
            return competingPop;
    }

    /**
     * creates a population of <tt>Chromosome</tt> to cross over
     * to the next generation.
     * @param population - population to crossOver
     * @return new population
     */
    private Population crossOver(Population population) {
            Population crossOverPop =
                            new Population(population.getAllChromosomes().length);

            for(int i = 0; i < noOfEliteGenes; i++) {

                    crossOverPop.getAllChromosomes()[i]
                                    = population.getAllChromosomes()[i];
            }
            for(int i = noOfEliteGenes; i < population.getAllChromosomes().length; i++) {
                    Chromosome chromosomeX =

    selectSurivingGenetics(population).getAllChromosomes()[0];
				Chromosome chromosomeY =

		selectSurivingGenetics(population).getAllChromosomes()[0];

				crossOverPop.getAllChromosomes()[i]=
						reproduce(chromosomeX, chromosomeY);
			}
			return crossOverPop;
		}

		/**
		 * Randomly crosses over genetics of two <tt>Chromosomes</tt>
		 * @param chromosomeX - first pool to select from
		 * @param chromosomeY - second pool to select from
		 * @return
		 */
		private Chromosome reproduce(Chromosome chromosomeX, Chromosome chromosomeY) {
			Chromosome chromosomeToCross = new Chromosome(targetGeneCode.length);

			for(int i = 0; i < chromosomeX.getGenes().length; i++) {
				if(Math.random() < 0.5) {
					chromosomeToCross.getGenes()[i] = chromosomeX.getGenes()[i];
				}
				else {
					chromosomeToCross.getGenes()[i] = chromosomeY.getGenes()[i];
				}
			}
			return chromosomeToCross;
		}

	} // end of class Chromosome.java
```

Population.java

```
import java.util.ArrayList;
import java.util.Arrays;

/**
 * Population holding <tt>Chromosome</tt> objects
 * creates a intial population
 * @author Luke McCann
 * @date December 2016
 *
 */
public class Population {

	private Chromosome[] dna; // the genetic code

	public Population(int length) {
		dna = new Chromosome[length];
	}

	/**
	 * initializes the population by
	 * adding new <tt>Chromosome</tt>  to each entry of
	 * the <tt>dna</tt> array
	 * essentially fills our <tt>dna</tt>
	 * object with genetic materials
	 * @return this Population object
	 */
	public Population initialize() {
		for(int i = 0; i < dna.length; i++) {
			dna[i] = new Chromosome(
					Algorithm.targetGeneCode.length).initialize(); //
initializes Chromosome with genes
		}
		sortByFitness(); // sorts the <tt>Chromosome</tt> by accuracy to the target
		return this;
	}

	/**
	 * sort the <tt>Chromosomes</tt> by genealogical fitness
	 * uses a lambda expression to return all <tt>dna</tt> objects
	 * that meet (condition) uses "pointer" to mark items
```

```
         * @return pointer - the pointers value
         */
        public void sortByFitness() {
                Arrays.sort(dna, (dnaX, dnaY) -> {   // -> is a new lambda expression in Java 8
                        int pointer = 0;                  // means that from <tt>dna</tt> array
we are returning
                                                        // list of <tt>chromosomes</tt> that
meet (condition)
                        if(dnaX.getFitness() > dnaY.getFitness()) {
                                pointer--;
                        }
                        else if (dnaX.getFitness() < dnaY.getFitness()) {
                                pointer++;
                        }
                        return pointer;

                });
        }

        // Getters and Setters
        public Chromosome[] getAllChromosomes() {
                return dna;
        }

} // end of class Populationjava
```

Algorithm.java

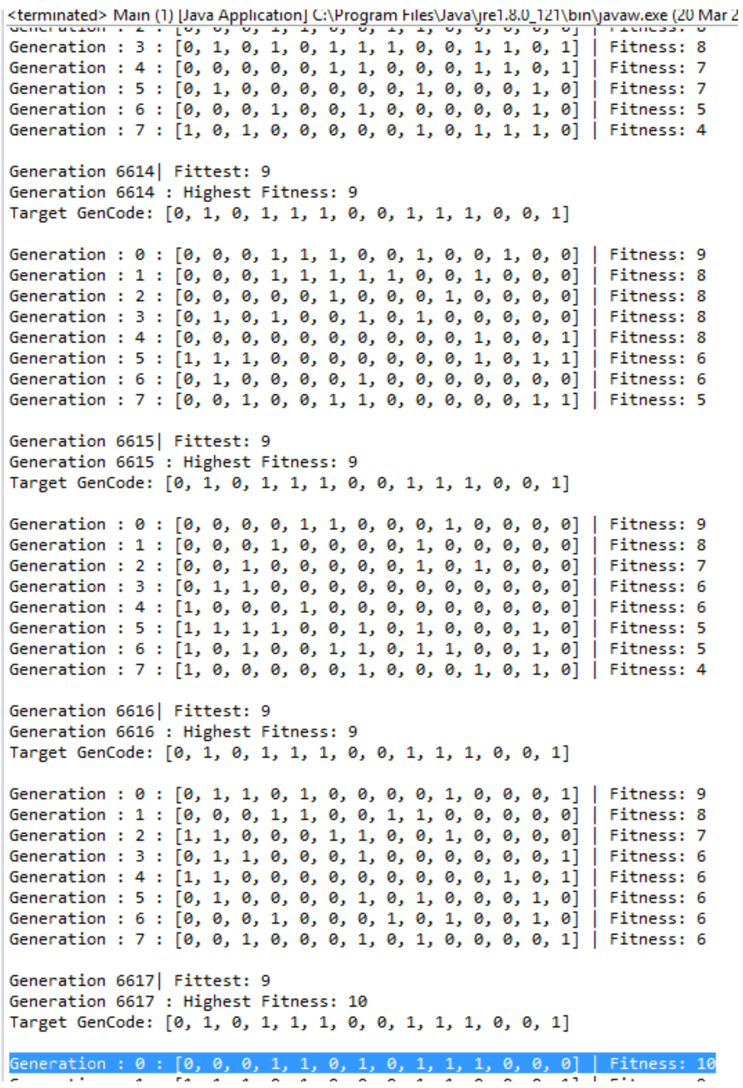
```
/**
 * Holds the main functionality of the GeneticAlgorithm
 * such as evoloution, mutation, reproduction
 * @author Luke McCann
 * @date December 2016
 *
 */
public class Algorithm {

        public static int[] targetGeneCode = {0,1,0,1,1,1,0,0,1,1,1,0,0,1};
        public static int population = 8;
        public static int noOfEliteGenes = 0;

        public static int maxSurvivors = 4;
        private static double mutationRate = 0.5;  // percentage expressed as double value

        public Algorithm() {
        }

        public Algorithm(int populationDensity, int[] targetGeneticCode) {
                Algorithm.population = populationDensity;
                Algorithm.targetGeneCode = targetGeneticCode;
        }

        /**
         * evolves the population
         * @param population - population to mutate
         * @return mutatedPopulation
         */
        public Population evolve(Population population) {
                return mutateAllPop(crossOver(population));
        }

        /**
         * Applies an element of randomness specified by the
         * mutationRate, needed in order for an answer to evolve
         * @param population - population to mutate
         * @return mutantChromosome - the <tt>Chromosome</tt> which has been mutated
         */
        private Chromosome mutateChromosome(Chromosome x) {
                Chromosome mutantChromosome =
                                new Chromosome(targetGeneCode.length);

                for(int i = 0; i < x.getGenes().length; i++) {
                        if(Math.random() < mutationRate) {
                                if(Math.random() < 0.5) {
                                        mutantChromosome.getGenes()[i] = 1;
                                }
                                else if(Math.random() > 0.5){
                                        mutantChromosome.getGenes()[i] = 0;
                                }
                                else {
                                        mutantChromosome.getGenes()[i] = x.getGenes()[i];
                                }
                        }
                }
                return mutantChromosome;
        }

        /**
         * creates a new population to mutate
         * @param population - population to mutate
         * @return mutatePop - mutated population
         */
        private Population mutateAllPop(Population population) {
                Population mutatePop =
                                new Population(population.getAllChromosomes().length);

                for(int i = 0; 0 < noOfEliteGenes; i++) {

                        mutatePop.getAllChromosomes()[i] =
                                        population.getAllChromosomes()[i]; //
                }
                for(int i = noOfEliteGenes; i <
                                population.getAllChromosomes().length; i++) {
                        mutatePop.getAllChromosomes()[i] =
mutateChromosome(population.getAllChromosomes()[i]);
                }
                return mutatePop;
        }


        /**
         * Randomly selects <tt>Chromosomes</tt>
         * to reuse: only those with the highest fitness
         * survive
         * @return competingPop - the population competing for highest fitness
         */
        private Population selectSurivingGenetics(Population population) {
                Population competingPop =
                                new Population(maxSurvivors);

                for(int i = 0; i < maxSurvivors ; i++) {
                        competingPop.getAllChromosomes()[i] =
                                        population.getAllChromosomes()

        [(int)(Math.random()*population.getAllChromosomes().length)];
                }
                competingPop.sortByFitness();
                return competingPop;
        }

        /**
         * creates a population of <tt>Chromosome</tt> to cross over
         * to the next generation.
         * @param population - population to crossOver
         * @return new population
         */
        private Population crossOver(Population population) {
                Population crossOverPop =
                                new Population(population.getAllChromosomes().length);

                for(int i = 0; i < noOfEliteGenes; i++) {

                        crossOverPop.getAllChromosomes()[i]
                                        = population.getAllChromosomes()[i];
                }
                for(int i = noOfEliteGenes; i < population.getAllChromosomes().length; i++) {
                        Chromosome chromosomeX =

        selectSurivingGenetics(population).getAllChromosomes()[0];

                        Chromosome chromosomeY =
        selectSurivingGenetics(population).getAllChromosomes()[0];

                        crossOverPop.getAllChromosomes()[i]=
                                        reproduce(chromosomeX, chromosomeY);
                }
                return crossOverPop;
        }

        /**
         * Randomly crosses over genetics of two <tt>Chromosomes</tt>
         * @param chromosomeX - first pool to select from
         * @param chromosomeY - second pool to select from
         * @return
         */
        private Chromosome reproduce(Chromosome chromosomeX, Chromosome chromosomeY) {
                Chromosome chromosomeToCross = new Chromosome(targetGeneCode.length);

                for(int i = 0; i < chromosomeX.getGenes().length; i++) {
                        if(Math.random() < 0.5) {
                                chromosomeToCross.getGenes()[i] = chromosomeX.getGenes()[i];
                        }
                        else {
                                chromosomeToCross.getGenes()[i] = chromosomeY.getGenes()[i];
                        }
                }
                return chromosomeToCross;
        }

} // end of class Algorithm
```

Main.java

```
import java.util.Arrays;

public class Main {

        private static int generation = 0;

        public static void main(String[] args) {
                Population initialPop = new Population(
                                Algorithm.population).initialize();

                Algorithm geneticAlgorithm = new Algorithm();

                while(initialPop.getAllChromosomes()[0].getFitness()
                                < Algorithm.targetGeneCode.length) {
                        generation++;

                        System.out.println("");
                        System.out.println(("Generation " + generation + "| Fittest: "
                                        + initialPop.getAllChromosomes()[0].getFitness()));

                        initialPop = geneticAlgorithm.evolve(initialPop);
                        initialPop.sortByFitness();

                        System.out.println("Generation " + generation + " : " + "Highest
Fitness: "
                                        + initialPop.getAllChromosomes()[0].getFitness());
                        print(initialPop, "Target GenCode: " +
Arrays.toString(Algorithm.targetGeneCode));
                }
        }

        public static void print(Population population, String target) {
                System.out.println(target);
                System.out.println("");

                for(int i = 0; i < population.getAllChromosomes().length; i++) {
                        System.out.println("Generation : " + i + " : " +

        Arrays.toString(population.getAllChromosomes()[i].getGenes()) +
                                                      " | Fitness: " +
population.getAllChromosomes()[i].getFitness());
```

```
                    }
          }
}
```

Test



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (20 Mar 2
Generation : 3 : [0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1] | Fitness: 8
Generation : 4 : [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1] | Fitness: 7
Generation : 5 : [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0] | Fitness: 7
Generation : 6 : [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0] | Fitness: 5
Generation : 7 : [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0] | Fitness: 4

Generation 6614| Fittest: 9
Generation 6614 : Highest Fitness: 9
Target GenCode: [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]

Generation : 0 : [0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0] | Fitness: 9
Generation : 1 : [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0] | Fitness: 8
Generation : 2 : [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0] | Fitness: 8
Generation : 3 : [0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0] | Fitness: 8
Generation : 4 : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1] | Fitness: 8
Generation : 5 : [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1] | Fitness: 6
Generation : 6 : [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] | Fitness: 6
Generation : 7 : [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1] | Fitness: 5

Generation 6615| Fittest: 9
Generation 6615 : Highest Fitness: 9
Target GenCode: [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]

Generation : 0 : [0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0] | Fitness: 9
Generation : 1 : [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] | Fitness: 8
Generation : 2 : [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0] | Fitness: 7
Generation : 3 : [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] | Fitness: 6
Generation : 4 : [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] | Fitness: 6
Generation : 5 : [1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0] | Fitness: 5
Generation : 6 : [1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0] | Fitness: 5
Generation : 7 : [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0] | Fitness: 4

Generation 6616| Fittest: 9
Generation 6616 : Highest Fitness: 9
Target GenCode: [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]

Generation : 0 : [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1] | Fitness: 9
Generation : 1 : [0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0] | Fitness: 8
Generation : 2 : [1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0] | Fitness: 7
Generation : 3 : [0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1] | Fitness: 6
Generation : 4 : [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1] | Fitness: 6
Generation : 5 : [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0] | Fitness: 6
Generation : 6 : [0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0] | Fitness: 6
Generation : 7 : [0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1] | Fitness: 6

Generation 6617| Fittest: 9
Generation 6617 : Highest Fitness: 10
Target GenCode: [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]

Generation : 0 : [0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0] | Fitness: 10
```

Self Assessments

8.1.1 Week 7

Week	Overall	Docs	Struct.	Names	Tests	Funct
1 & 2 Search timer	A	B	A	A	C	A
3 & 4 Generic swap	A	B	A	A	C	A
5 Sorting	A	A	A	B	B	C
6 Linked lists	B	C	B	B	B	C
8 Binary trees	B	B	A	B	C	A

Evidence/justification:

As the first part of the term I have found these tutorials a mix of challenging and lesser challenging items. For the first three exercises I feel I had an excellent overall understanding in very little time, as I managed to implement all of the solutions to the problem statements relatively swiftly and simply.

I feel I have understood the Search Timer and Generic Swap best. When implementing these I had relatively not issues, however I do feel I could have progressed in terms of testing the items in a better format and with more data types.

This is mostly due to a lack of practice with writing test cases in the past, being relatively new to programming I have struggled in some aspects more than others, and some of the more complex algorithms have taken up more time to implement (for instance the Binary Tree).

Whilst working on these exercises I believe I have improved greatly on my knowledge of Junit testing, as well as expanding my understanding of Java, what it has to offer and most importantly how many of the items which are already implemented for users to use work. I believe this is important to understand as understanding how these pre-coded libraries and classes work it gives the user more control over how they can manipulate the items within them.

Although I have tried my best to structure my code to make it readable, some of the items could have possibly been structure in a better way. I have gotten into the habit of using another structuring scheme when writing code in Visual Studio, to what I do when I am writing in Eclipse, and this can mean I accidentally switch between the two at times without thinking, I do consciously attempt to reformat it and keep the code consistent, however it has slowed down development at times.

8.1.2 Consolidation week

Week	Overall	Docs	Struct.	Names	Tests	Funct
9 Hashtable	A	Not applicable. Think about how well you have Hashtable explained the hashtable's behaviour				
10 Depth first traversal	C (Revised) A	A	B	B	C	C (Revised) A
11 Reference Counting Topological Sort	B	C	B	A	C	B

Evidence/justification:

Overall I believe my explanation of Hashtables is as in depth as it can be, I have included screenshots from blueJay and an explanation as to why the functionality happens as it does, as well as the behind the scenes of how Hashtables work, and why this means they produce this "output".

I have recently updated my Overall for Depth First Traversal after completing the item again later on in the year. The reason for this is that although I have struggled to understand this implementation and how it was supposed to work I have done more research into the matter and, overall, with improving my skills and knowledge further with the recent weeks became able to tackle the issue fully and implement the functionality, the testing could have been more thorough, however I am unsure on how this item would be properly tested and therefore have only given myself a C for testing.

Although my understanding of these items on the first pass was a bit inconclusive I have tried my best to research more and improve myself to a point at which I now feel I understand these in much more detail. I am more confident in implementing these items now and have managed to overcome problems at which I would not have been at the beginning of the year.

8.1.3 Week 17

Week	Overall	Docs	Struct.	Names	Tests	Funct
13 Counter Behaviour	A	Not applicable. Think about how well you have Counter analysed the (possible) behaviour of the behaviour counters				
14 Dekker trains	(Revisited) B	B	B	A	C	B
15 Semaphore Behaviour	A	Not applicable. Think about how whether you have Semaphore identified and the problem, and about how behaviour well you have explained it				
16 Locks and Conditions	B	C	A	B	C	B

Evidence/justification:

During week13 I believe my understanding of the Counters functionality is shown well and I have provided a thorough description of the behaviour of the Counters both if they were to execute alone, and when they execute together. Atop of this I also included a variety of tests and examples to back up my explanations.

Furthermore I have also come back to Dekkers Algorithm in the more recent weeks up to the deadline, this is due to the fact that I struggled to understand the original implementation of the algorithm and found it hard to visualise inside my head. Instead I found it much better to come back to after attempting some more exercises, in which I believe I have now correctly implemented Dekkers Algorithm in the revised weeks, I was unsure about what the output should be, however now it seems to be fully functional, on the contrary, I was also unsure on how to go about testing such an algorithm, so instead ran the program a few times and documented the output, however there were no significant changes.

Week 16 I found to be possibly the most challenging of all of the weeks. In this I have struggled with Monitors for a few weeks, I found it helpful to re-watch the lectures and read through online tutorial to get to grips with the workings of the monitors. Upon attempting this item again I found it much easier to understand, and found the exercise to be easier than I had thought so in my head, due to this growth in understanding I have given my oveall understanding of this item a B, however, much like Dekker's Algorithm, testing for this item was unclear as I do not yet understand how to go about testing this item in Junit and therefore have given myself a C for testing.

8.1.4 Week 22

Week	Topic	Grade	Criteria
17	Modelling Circuits	A+	Have you derived a matrix for a half-adder? Have you correctly applied the matrix methods for constructing sequential and parallel circuits? Have you explained/justified/proved your derivation? Have you tested it on (matrix representations of) various inputs?
20	Quantum Computing	A+	Have you fully analysed the values that will appear at points A, B, and C in the circuit? Have you discussed the relationship between the values appearing at A and C? Have you considered what the implications of a purely probabilistic model would be for maintaining this relationship?

Evidence/justification:

For these two weeks I have given myself the highest grade of all of the weeks. During the time studying Quantum Computing I have put a lot of time and effort into learning the mathematics and thoroughly enjoyed the “Quantum” and “Physics” based sides of the module. I believe this area has taught me the most and helped me to become more confident in my abilities. I have completed many additional exercises for Quantum Computing and Modelling Circuits, including drawing out all of the truth tables in both original and Qubit formats in order to help with calculations.

I have documented all of the calculations used and worked out each item in long hand, as well as providing my additional learning notes of which I used to help myself learn the necessary skills in order to complete the tasks.

Ontop of the items in the logbook which I have documented, I have also read through the book “Quantum Computing For Computer Scientists”.

Overall I feel I have completed this task to a very high quality and believe it to be one of the best items I have completed in this term.

General Assessment

Assessment Criterion	Grade
Answers to flagged logbook questions	B+
Answers to other practical questions	B
Other practical work (additional exercises you have undertaken of your own initiative)	A
Understanding of the module material to date	A
Level of self-reflection & evaluation	B
Participation in timetabled activities	A
Time spent outside timetabled classes (guideline is two hours self-study for each hour of timetabled study)	A+

Evidence/justification:

My reasoning behind these grades is that I believe I have completed this logbook to a good quality standard. I have answered every logbook question of each week to the best of my abilities and tested and documented as much as my current skill level is capable of, on some of the weeks I have documented my answer to other practical questions, however this has not been possible for all of the weeks as my logbook has grown rather large, not only this but on some of the more challenging weeks I had focused purely on the logbook questions in order to imbed a clear understanding of the main goals, this is more apparent in the more complex questions in which I became stuck and had to gather more knowledge and experience in order to complete the question.

Atop of this I have done a lot of additional practical work from my own initiative, much of which is also documented in the logbook. This has included items from Quantum Computing mathematics in order to improve my maths skills to complete the questions and understand Quantum Circuits and Quantum Gates in greater detail, to creating a program at which I have given myself examples in order to gain an understanding of Big O Notation in the early weeks and finally to the more challenging aspects at which I implemented to try and improve my skills as a programmer including the Perlin Noise Algorithm and Genetic Algorithm. These implementations came from my own learning outside of university, as I have developed a keen interest in evolutionary computing.

Although I have struggled on areas of the module and become focused on one area for large amounts of time in order to understand and complete the task at hand, I believe this has improved my skills and confidence vastly. In saying this I not have a greater understanding of the items visited and have shown this in the "Revisit" part of my logbook. In this area I have added new solutions to older tasks of which I had already completed but did not fully understand, meaning that my code either did not execute correctly, or that my outputs were incorrect. In revisiting these items and fixing the previous bugs and errors I believe I have shown that I now have a much greater understanding of the module than I had previously (upon first implementing these).

I believe I have given a good self-analysis for each section, however often struggle with communication and grammar, this meaning it could possibly be improved.

Finally, I have participated in all of the timetabled activities and spent the majority of my own time outside of class studying and improving on my work, overall I have improved vastly from the beginning of the year in both skills and confidence and believe this has been reflected in my work.