**Table 1:**

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayBag** | **LinkedBag** |
| add (newEntry) | O(n) | O(1) |
| remove( ) | O(n) | O(1) |
| remove(newEntry) | O(n) | O(n) |
| clear( ) | O(1) | O(1) |
| getFrequency(anEntry) | O(n) | O(n) |
| contains(anEntry) | O(n) | O(n) |
| getCount( ) | O(1) | O(1) |
| isEmpty( ) | O(1) | O(1) |

**Table 2:**

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayBag** | **LinkedBag** |
| **add ( ) : 1000 times** | Run 1: 0 | Run 1: 0 |
| Run 2: 0 | Run 2: 16 |
| Run 3: 0 | Run 3: 0 |
| Average: 0 | Average: 5 1/3 |
| **add ( ) : 10,000 times** | Run 1: 0 | Run 1: 0 |
| Run 2: 0 | Run 2: 0 |
| Run 3: 0 | Run 3: 0 |
| Average: 0 | Average: 0 |
| **add ( ) : 100,000 times** | Run 1: 15 | Run 1: 16 |
|  | Run 2: 16 | Run 2: 15 |
|  | Run 3: 16 | Run 3: 16 |
|  | Average: 15 2/3 | Average: 15 2/3 |
| **add ( ) : 1,000,000 times** | Run 1: 32 | Run 1: 203 |
|  | Run 2: 31 | Run 2: 203 |
|  | Run 3: 31 | Run 3: 203 |
|  | Average: 31 1/3 | Average: 203 |

**Table 3:**

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayBag** | **LinkedBag** |
| **remove ( ) : 1000 times** | Run 1: 0 | Run 1: 0 |
| Run 2: 0 | Run 2: 0 |
| Run 3: 0 | Run 3: 0 |
| Average: 0 | Average: 0 |
| **remove ( ) : 10,000 times** | Run 1: 0 | Run 1: 0 |
| Run 2: 0 | Run 2: 0 |
| Run 3: 0 | Run 3: 0 |
| Average: 0 | Average: 0 |
| **remove ( ) : 100,000 times** | Run 1: 0 | Run 1: 0 |
|  | Run 2: 0 | Run 2: 0 |
|  | Run 3: 0 | Run 3: 0 |
|  | Average: 0 | Average: 0 |
| **remove ( ) : 1,000,000 times** | Run 1: 15 | Run 1: 0 |
|  | Run 2: 0 | Run 2: 0 |
|  | Run 3: 16 | Run 3: 0 |
|  | Average: 10 ½ | Average: 0 |

Plot Graphs:

The add operation takes roughly the same amount of time for both data structures up to 100,000 objects. After 100,000 the difference in efficiency between the two becomes more apparent. At one million players added it takes LinkedBag on average 203ms to complete the add operation, while it takes the ArrayBag only about 31ms to complete.

Much like the add operation the remove operation takes an insignificant amount of time for both data structures up to 100,000 objects. After 100,000 objects ArrayBag takes on average 10.5ms to complete, whereas LinkedBag takes 0ms to complete.

test program:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package lab5;

import java.util.Random;

/\*\*

\* Test class for analyzing the time taken in milliseconds for carrying out selected operations of the ArrayBag and LinkedBag classes.

\* @author Jacob Huesman

\*/

public class Test {

public static void main(String[] args){

//1. Create an ArrayBag object named teamArray using the default constructor to store Player objects

ArrayBag<Player> teamArray = new ArrayBag<>();

//2. Create a LinkedBag object named teamLinkedList to store Player ojbects

LinkedBag<Player> teamLinkedList = new LinkedBag<>();

//3. Create a long type of variable named start and set it's value to System.currentMilliseconds()

long start = System.currentTimeMillis();

//4. Populate the ArrayBag with 1000 Player objects

Random random = new Random();

for(int i=0; i<1000; i++){

teamArray.add(new Player("John","guard",random.nextInt()));

}

//5. Create a long type of variable named end and set it's value to System.currentMilliseconds()

long end = System.currentTimeMillis();

//6. Calculate elapsed time using the difference between start and end

long elapsed = end - start;

//7. Print the elapsed time for teamArray using verbose output

System.out.println("The time it took to add 1000 Player entries to teamArray is: " + elapsed);

//8. Reuse the start variable and set it's value to System.currentMilliseconds()

start = System.currentTimeMillis();

//9. Populate the LinkedBag with 1000 Player objects

for(int i=0; i<1000; i++){

teamLinkedList.add(new Player("John","guard",random.nextInt()));

}

//10. Reuse the end variable and set it's value to System.currentMilliseconds()

end = System.currentTimeMillis();

//11. Calculate elapsed time using the difference between start and end

elapsed = end - start;

//12. Print the elapsed time for teamLinkedList using verbose output

System.out.println("The time it took to add 1000 Player entries to teamLinkedList is: " + elapsed);

//13. Print the count of teamArray and teamLinkedList on the output window

System.out.println("The teamArray count is: " + teamArray.getCurrentSize());

System.out.println("The teamLinkedList count is: " + teamArray.getCurrentSize());

//14. Reuse the start variable and set it's value to System.currentMilliseconds()

start = System.currentTimeMillis();

//15. Retrieve the Player object in the index position 0 of teamArray

Player retrieved = teamArray.get(0);

//16. Remove the Player object retrieved in the earlier step

teamArray.remove(retrieved);

//17. Reuse the end variable and set it's value to System.currentMilliseconds()

end = System.currentTimeMillis();

//18. Calculate elapsed time using the difference between start and end

elapsed = end - start;

//19. Print the elapsed time for teamArray using verbose output

System.out.println("The time it took to remove 1 Player entry from index 0 in teamArray is: " + elapsed);

//20. Repeat steps 14 to 19 for teamLinkedList

//Start time

start = System.currentTimeMillis();

//Retrieve Player

retrieved = teamLinkedList.getItem(0);

//Remove Player

teamLinkedList.remove(retrieved);

//End time

end = System.currentTimeMillis();

//Calculate elapsed time using the difference between start and end

elapsed = end - start;

//Print the elapsed time for teamArray using verbose output

System.out.println("The time it took to remove 1 Player entry from index 0 in teamLinkedList is: " + elapsed);

}

}

output 10,000:

run:

The time it took to add 10,000 Player entries to teamArray is: 0

The time it took to add 10,000 Player entries to teamLinkedList is: 0

The teamArray count is: 10000

The teamLinkedList count is: 10000

The time it took to remove 1 Player entry from index 0 in teamArray is: 15

The time it took to remove 1 Player entry from index 0 in teamLinkedList is: 0

BUILD SUCCESSFUL (total time: 0 seconds)

output 1,000,000:

run:

The time it took to add 1,000,000 Player entries to teamArray is: 47

The time it took to add 1,000,000 Player entries to teamLinkedList is: 203

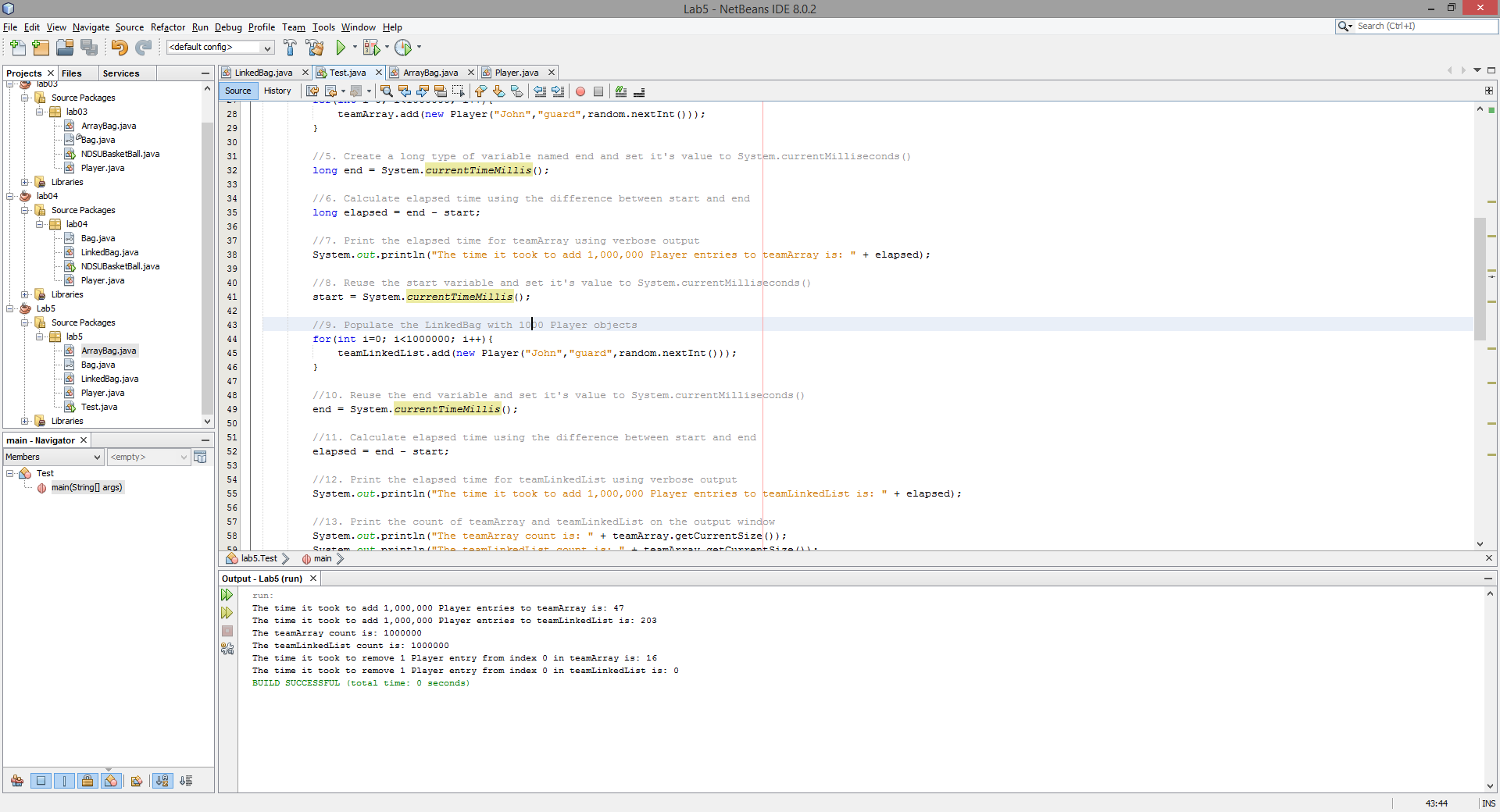
The teamArray count is: 1000000

The teamLinkedList count is: 1000000

The time it took to remove 1 Player entry from index 0 in teamArray is: 16

The time it took to remove 1 Player entry from index 0 in teamLinkedList is: 0

BUILD SUCCESSFUL (total time: 0 seconds)



UML Diagram:

