**CSC 151 Assignment #8**

1. **Honor Code**
2. *For individual assignments: Jane Doe and John Doe will be replaced by your full name(s)*

*I affirm that I have carried out my academic endeavors with full academic honesty.*

*[Signed, Manav Bilakhia]*

1. Resources/References

Geeksforgeeks for syntax

1. **Java files and outputs**
2. Java files

Class: ListItem

/\*  
 \* I affirm that I have carried out my academic endeavors with full academic honesty.  
 \*/  
package assignment;  
  
import java.util.Objects;  
  
*/\*\*  
 \* The class for an integer and frequency pair  
 \** ***@author*** *Zeynep Orhan  
 \*  
 \*/*public class ListItem {  
 //Instance variables int item and int freq  
 private int item;  
 private int freq;  
//Constructors: One with no parameters and one with 2 integer parameters  
 public ListItem()  
 {  
 this.item = 0;  
 this.freq = 0;  
 }  
 public ListItem(int item, int freq)  
 {  
 this.setItem(item);  
 this.setFreq(freq);  
 }  
//setters and getters  
  
 public int getItem() {  
 return item;  
 }  
  
 public int getFreq() {  
 return freq;  
 }  
  
 public void setItem(int item) {  
 this.item = item;  
 }  
  
 public void setFreq(int freq) {  
 this.freq = freq;  
 }  
  
 //toString  
 public String toString() {  
 return "[" + item + ", " + freq + "]";  
 }  
  
//hashCode  
  
 //hashCode  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + freq;  
 result = prime \* result + item;  
 return result;  
 }  
  
 //equals  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 ListItem other = (ListItem) obj;  
 if (freq != other.freq)  
 return false;  
 if (item != other.item)  
 return false;  
 return true;  
 }  
  
 public static void main(String[] args) {  
 ListItem lI1 = new ListItem(1,1);  
 ListItem lI2 = new ListItem(3,5);  
 ListItem lI3 = new ListItem(8,4);  
 System.*out*.println(lI1);  
 System.*out*.println(lI2);  
 System.*out*.println(lI3);  
 }  
}

Class LikstItemTest

package assignment;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class ListItemTest {  
  
 @org.junit.jupiter.api.Test  
 void getItem() {  
 ListItem item = new ListItem(2,5);  
 *assertEquals*(2,item.getItem());  
 }  
  
 @org.junit.jupiter.api.Test  
 void getFreq() {  
 ListItem item = new ListItem(2,5);  
 *assertEquals*(5,item.getFreq());  
 }  
  
 @org.junit.jupiter.api.Test  
 void testEquals() {  
 ListItem item1 = new ListItem(2,5);  
 ListItem item2 = new ListItem(3,4);  
 ListItem item3 = new ListItem(2,5);  
 *assertTrue*(item3.equals(item1));  
 *assertFalse*(item1.equals(item2));  
 }  
}

Class: ListEncodeDecode

package assignment;  
import java.util.\*;  
*/\*\*  
 \* This class packs/unpacks the lists of integers  
 \* The repeated items are represented  
 \* by the item and its frequency in the packed version  
 \*  
 \* The items and the frequencies can be unpacked  
 \* as the full representation  
 \*  
 \** ***@author*** *Zeynep Orhan  
 \*  
 \*/*public class ListEncodeDecode {  
*/\*\*  
 \* Recursive static method packed: Pack the list of repeated sequences into item and frequency pairs  
 \*  
 \** ***@param*** *repeatedList an ArrayList of Integers of repeated sequences  
 \** ***@param*** *packList an ArrayList of ListItem where we have item and frequencies  
 \** ***@param*** *start index of the starting position in the repeatedList  
 \*  
 \*/* public static void packed(ArrayList <Integer>repeatedList, ArrayList <ListItem>packList,int start)  
 {  
 Collections.*sort*(repeatedList);  
 if(start >= repeatedList.size())  
 {  
 return;  
 }  
 if(packList.isEmpty())  
 {  
 ListItem item = new ListItem(repeatedList.get(start),1);  
 packList.add(item);  
  
 }  
 else {  
 ListItem current = packList.get(packList.size()-1);  
 if (repeatedList.get(start)== current.getItem())  
 {  
 current.setFreq(current.getFreq()+1);  
 }  
 else{  
 ListItem item = new ListItem(repeatedList.get(start),1);  
 packList.add(item);  
 }  
 }  
 *packed*(repeatedList, packList, start+1);  
 }  
  
*/\*\*  
 \* Recursive static method unpacked: Unpack the list of item and frequency pairs into repeated sequences  
 \*  
 \* This method mutates the elements of the repeated list  
 \** ***@param*** *packList an ArrayList of ListItem where we have item and frequencies  
 \** ***@param*** *repeatedList an ArrayList of Integers of repeated sequences  
 \** ***@param*** *start index of the starting position in the repeatedList  
\*/*public static void unpacked(ArrayList <ListItem>packList,ArrayList <Integer>repeatedList,int start)  
{  
  
 if(start >= packList.size())  
 {  
 return;  
 }  
 ListItem current = packList.get(start);  
 repeatedList.add(current.getItem());  
 current.setFreq(current.getFreq()-1);  
 if(current.getFreq()==0)  
 {  
 start+=1;  
 }  
 *unpacked*(packList,repeatedList,start);  
}  
 */\*\*  
 \* Recursive static method unpackItem: Unpack one item, add item to repeatedList howMany times  
 \** ***@param*** *item int item to be added  
 \** ***@param*** *howMany int frequency of the item  
 \** ***@param*** *repeatedList ArrayList<Integer> the result of adding item howMany times to this list  
\*/* public static void unpackItem(int item, int howMany, ArrayList <Integer>repeatedList)  
 {  
 if (howMany<=0)  
 return;  
 repeatedList.add(item);  
 *unpackItem*(item, howMany-1,repeatedList);  
 }  
  
 public static void main(String[] args) {  
 //Integer items[] = {2};  
 Integer items[] = {1,1,1,2,3,3,3,3,3,4,4};  
//Integer items[] = { 1, 1, 1 };  
 ArrayList<Integer> uncompressed = new ArrayList<>();  
 ArrayList<ListItem> compressed = new ArrayList<>();  
 ArrayList<Integer> newUncompressed = new ArrayList<>();  
 ArrayList<Integer> repeatedList = new ArrayList<>();  
 Collections.*addAll*(uncompressed, items);  
 *packed*(uncompressed, compressed, 0);  
 *unpacked*(compressed, newUncompressed, 0);  
 *unpackItem*(5,7,repeatedList);  
 System.*out*.println(uncompressed);  
 System.*out*.println(compressed);  
 System.*out*.println(newUncompressed);  
 System.*out*.println(repeatedList);  
 }  
}

Class: ListEncodeDecodeTest

package assignment;  
  
import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
import java.util.Collections;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class ListEncodeDecodeTest {  
  
 @Test  
 void packed() {  
 Integer items[] = {1,1,1,3,3,3,3,3,4,4};  
 ArrayList<Integer> uncompressed = new ArrayList<>();  
 ArrayList<ListItem> compressedexp = new ArrayList<>();  
 ArrayList<ListItem> compressedact = new ArrayList<>();  
 Collections.*addAll*(uncompressed, items);  
 ListItem item1 = new ListItem(1,3);  
 ListItem item2 = new ListItem(3,5);  
 ListItem item3 = new ListItem(4,2);  
 compressedexp.add(item1);  
 compressedexp.add(item2);  
 compressedexp.add(item3);  
 ListEncodeDecode.*packed*(uncompressed, compressedact, 0);  
 *assertArrayEquals*(compressedexp.toArray(),compressedact.toArray());  
  
 }  
  
 @Test  
 void unpacked() {  
 Integer items[] = {1,1,1,3,3,3,3,3,4,4};  
 ArrayList<Integer> uncompressedexp = new ArrayList<>();  
 ArrayList<Integer> uncompressedact = new ArrayList<>();  
 ArrayList<ListItem> compressed = new ArrayList<>();  
 Collections.*addAll*(uncompressedexp, items);  
 ListItem item1 = new ListItem(1,3);  
 ListItem item2 = new ListItem(3,5);  
 ListItem item3 = new ListItem(4,2);  
 compressed.add(item1);  
 compressed.add(item2);  
 compressed.add(item3);  
 ListEncodeDecode.*unpacked*(compressed,uncompressedact,0);  
 *assertArrayEquals*(uncompressedexp.toArray(),uncompressedact.toArray());  
  
 }  
  
 @Test  
 void unpackItem() {  
 Integer items[] = {5,5,5,5,5,5,5};  
 ArrayList<Integer> repeatedListexp = new ArrayList<>();  
 ArrayList<Integer> repeatedListact = new ArrayList<>();  
 Collections.*addAll*(repeatedListexp,items);  
 ListEncodeDecode.*unpackItem*(5,7,repeatedListact);  
 *assertArrayEquals*(repeatedListexp.toArray(),repeatedListact.toArray());  
  
  
 }  
}

1. Sample output 1
2. Describe your test 1: See if the packed method works
3. Text output 1:

[1, 1, 1, 2, 3, 3, 3, 3, 3, 4, 4]

[[1, 3], [2, 1], [3, 5], [4, 2]]

1. Screenshot 1:

A picture containing text, black, watch, meter

Description automatically generated

1. Sample output 2
2. Describe your test 2: See if the packed method works
3. Text output 2:

[[1, 3], [2, 1], [3, 5], [4, 2]]

[1, 1, 1, 2, 3, 3, 3, 3, 3, 4, 4]

1. Screenshot 2:

A picture containing text, watch, sign

Description automatically generated

1. Sample output 3
2. Describe your test 3: See if the unpackItem method works
3. Text output 3:

[5, 5, 5, 5, 5, 5, 5]

1. Screenshot 3:

**Logo

Description automatically generated**