

Jul 11, 02 13:18

Card.java

Page 1/1

```
public class Card implements Comparable
{
    // other methods and data fields not shown

    /**
     * Return a value according to whether
     * this Card is greater, equal to, or less than
     * the card passed in as parameter o. The value
     * returned is consistent with the specifications
     * of the Comparable interface's compareTo method.
     */
    public int compareTo(Object o)
    {
        // implementation not shown
        return 1;
    }
}
```

Jul 11, 02 13:40

CodeTree.java

Page 1/2

```

public class CodeTree
{
    /**
     * Construct a codetree.
     */
    public CodeTree()
    {
        myRoot = new TreeNode("",
            new TreeNode("",
                new TreeNode("s",null,null),
                new TreeNode("l",
                    new TreeNode("u",null,null),
                    new TreeNode("y",null,null))),
            new TreeNode("n",
                new TreeNode("n",null,null),
                new TreeNode("i",null,null)));
    }

    /**
     * Returns a decoded word for code.
     * @precondition code valid string of 0's and 1's
     * @return the decoded word for code
     */
    public String bitsToWord(String code)
    {
        TreeNode current = myRoot;
        String result = "";

        for(int k=0; k < code.length(); k++) {
            if (code.substring(k,k+1).equals("0")) {
                current = current.getLeft();
            }
            else {
                current = current.getRight();
            }
            if (current.getLeft() == null &&
                current.getRight() == null) { // it's a leaf?
                result += current.getValue();
                current = myRoot;
            }
        }
        return result;
    }

    /**
     * Returns the code for word
     * @precondition each character of word is in a leaf of codetree
     * @return the code for word
     */
    public String wordToBits(String word)
    {
        String result = "";
        for(int k=0; k < word.length(); k++) {
            result += wordToBitsHelper(word.substring(k,k+1),myRoot,"");
        }
        return result;
    }

    /**
     * Returns the code for s in codetree, returns "" if s not in tree
     * @precondition pathSoFar is path of 0's and 1's from myRoot to t
     * @return the path from myRoot to leaf containing s ,"" if no path
     */
    private String wordToBitsHelper(String s, TreeNode t,

```

Jul 11, 02 13:40

CodeTree.java

Page 2/2

```

        String pathSoFar)
    {
        if (t.getLeft() == null && t.getRight() == null) { // leaf?
            if (t.getValue().equals(s)) {
                return pathSoFar;
            }
            else {
                return "";
            }
        }
        String result = wordToBitsHelper(s,t.getLeft(),
            pathSoFar+"0");
        if (result.length() > 0) {
            return result;
        }
        return wordToBitsHelper(s,t.getRight(),pathSoFar+"1");
    }

    private TreeNode myRoot;

    public static void main(String[] args)
    {
        CodeTree ct = new CodeTree();
        System.out.println(ct.bitsToWord("1110"));
        System.out.println(ct.bitsToWord("000101010011"));
        System.out.println();
        System.out.println(ct.wordToBits("in"));
        System.out.println(ct.wordToBits("sunny"));
    }
}

```

Jul 11, 02 13:35

Compete.java

Page 1/1

```

import java.util.*;

public class Compete
{
    private Queue myDiscards;

    /**
     * Construct with empty discard pile
     */

    public Compete()
    {
        myDiscards = new ListQueue();
    }

    /**
     * precondition: myDiscards is empty
     * Simulate one round of Compete.
     */

    public void oneRound(Queue pile1, Queue pile2)
    {
        while (true) {
            if (pile1.isEmpty() && pile2.isEmpty()) {
                break;
            }
            else if (pile1.isEmpty()) {
                appendQueue(pile2, myDiscards);
                break;
            }
            else if (pile2.isEmpty()) {
                appendQueue(pile1, myDiscards);
                break;
            }

            Card c1 = (Card) pile1.dequeue();
            Card c2 = (Card) pile2.dequeue();

            if (c1.compareTo(c2) == 0) {
                myDiscards.enqueue(c1);
                myDiscards.enqueue(c2);
            }
            else if (c1.compareTo(c2) < 0) {
                appendQueue(pile2, myDiscards);
                pile2.enqueue(c1);
                pile2.enqueue(c2);
                break;
            }
            else {
                appendQueue(pile1, myDiscards);
                pile1.enqueue(c1);
                pile2.enqueue(c2);
                break;
            }
        }
    }

    /**
     * Remove all objects from source and put them
     * in destination in the order in which they're removed.
     */
    private void appendQueue(Queue destination, Queue source)
    {
        while (! source.isEmpty()) {
            destination.enqueue(source.dequeue());
        }
    }
}

```

Jul 10, 02 15:22

GroceryItem.java

Page 1/1

```
public interface GroceryItem
{
    String getName();    // returns name of item
    double getPrice();  // returns price of item
    int    getSize();   // returns #ounces (or size) of item
    String getCategory(); // returns category of item
}
```

Jul 10, 02 15:22 **GroceryStore.java** Page 1/2

```

import java.util.ArrayList;
import java.util.Iterator;

public class GroceryStore
{
    ArrayList myItems;

    /**
     * Creates an initially empty grocery store
     */
    public GroceryStore()
    {
        myItems = new ArrayList();
    }

    /**
     * Changes the price of the item associated with itemName
     */
    public void setPrice(String itemName, double price)
    {
        GroceryItem item = getItem(itemName);
        if (item != null) {
            ((GItem) item).price = price;
        }
    }

    /**
     * returns the item associated with itemName
     */
    public GroceryItem getItem(String itemName)
    {
        return getGItem(itemName);
    }

    /**
     * returns a (possibly empty) ArrayList of all
     * the items in the specified category
     */
    public ArrayList getAllItems(String category)
    {
        ArrayList list = new ArrayList();
        for(int k=0; k < myItems.size(); k++) {
            GItem item = (GItem) myItems.get(k);
            if (item.getCategory().equals(category)) {
                list.add(item);
            }
        }
        return list;
    }

    /**
     * precondition: names.length == prices.length
     * changes the price of the item associated with names[k]
     * to the price specified by prices[k]
     */
    public void changePrices(String[] names, double[] prices)
    {
        for(int k=0; k < names.length; k++) {
            setPrice(names[k], prices[k]);
        }
    }

    private GItem getGItem(String itemName)

```

Jul 10, 02 15:22 **GroceryStore.java** Page 2/2

```

    {
        Iterator it = myItems.iterator();
        while (it.hasNext()) {
            GItem gitem = (GItem) it.next();
            if (gitem.getName().equals(itemName)) {
                return gitem;
            }
        }
        return null;
    }

    private class GItem implements GroceryItem
    {
        String name;
        double price;
        String category;
        int size;
        public String getName() { return name; }
        public double getPrice() { return price; }
        public int getSize() { return size; }
        public String getCategory() { return category; }
    }
}

```

Jul 11, 02 13:02

ListQueue.java

Page 1/1

```
import java.util.LinkedList;

public class ListQueue implements Queue
{
    public ListQueue()
    {
        list = new LinkedList();
    }

    public void enqueue(Object x)
    {
        list.addLast(x);
    }

    public Object dequeue()
    {
        return list.removeFirst();
    }

    public Object peekFront()
    {
        return list.getFirst();
    }

    public boolean isEmpty()
    {
        return list.size() == 0;
    }

    private LinkedList list;
}
```

Jul 11, 02 12:59

Queue.java

Page 1/1

```
public interface Queue
{
    // precondition: returns true if queue is empty, false otherwise
    boolean isEmpty();

    // precondition: queue is [e1, e2, ..., en] with n >= 0
    // postcondition: queue is [e1, e2, ..., en, x]
    void enqueue(Object x);

    // precondition: queue is [e1, e2, ..., en] with n >= 1
    // postcondition: queue is [e2, ..., en]; returns e1
    // throws an unchecked exception if the queue is empty
    Object dequeue();

    // precondition: queue is [e1, e2, ..., en] with n >= 1
    // postcondition: returns e1
    // throws an unchecked exception if the queue is empty
    Object peekFront();
}
```

Jul 10, 02 15:09

Stats.java

Page 1/2

```

import java.util.ArrayList;

public class Stats
{
    /**
     * precondition: nums.length > 0
     * postcondition: returns the maximal value in nums
     */

    private static int findMax(int[] nums)
    {
        int max = nums[0];

        for(int k=1; k < nums.length; k++) {
            if (nums[k] > max) {
                max = nums[k];
            }
        }
        return max;
    }

    /**
     * precondition: nums.size() > 0; nums contains Integer objects
     * postcondition: returns the maximal value in nums
     */

    private static Integer findMax(ArrayList nums)
    {
        Integer max = (Integer) nums.get(0);
        for(int k=1; k < nums.size(); k++) {
            Integer i = (Integer) nums.get(k);
            if (i.compareTo(max) > 0) {
                max = i;
            }
        }
        return max;
    }

    /**
     * precondition: tally.length > 0
     * postcondition: returns an int array that contains the modes(s);
     *                 the array's length equals the number of modes.
     */

    public static int[] calculateModes(int[] tally)
    {
        int count = 0;
        int max = findMax(tally);
        for(int k=0; k < tally.length; k++) {
            if (tally[k] == max) {
                count++;
            }
        }

        count = 0;
        int[] retval = new int[count];
        for(int k=0; k < tally.length; k++) {
            if (tally[k] == max) {
                retval[count] = k;
                count++;
            }
        }
        return retval;
    }
}

```

Jul 10, 02 15:09

Stats.java

Page 2/2

```

/**
 * precondition: tally.size() > 0; tally contains Integer objects
 * postcondition: returns an ArrayList that contains the modes(s);
 *                 the ArrayList's size equals the number of modes.
 */

public static ArrayList calculateModes(ArrayList tally)
{
    ArrayList retval = new ArrayList();
    Integer max = findMax(tally);
    for(int k=0; k < tally.size(); k++) {
        if (tally.get(k).equals(max)) {
            retval.add(tally.get(k));
        }
    }
    return retval;
}

/**
 * precondition: tally.length > 0;
 *                 0 < k <= total number of values in data collection
 * postcondition: returns the kth value in the data collection
 *                 represented by tally
 */

public static int kthDataValue(int[] tally, int k)
{
    int count = 0;
    for(int j=0; j < tally.length; j++) {
        count += tally[j];
        if (count >= k) {
            return j;
        }
    }
    // error: should never reach here
    return tally.length-1;
}

/**
 * precondition: tally.size() > 0; tally contains Integer objects
 *                 0 < k <= total number of values in data collection
 * postcondition: returns the kth value in the data collection
 *                 represented by tally
 */

public static int kthDataValue(ArrayList tally, int k)
{
    int count = 0;
    for(int j=0; j < tally.size(); j++) {
        Integer i = (Integer) tally.get(j);
        count += i.intValue();
        if (count >= k) {
            return j;
        }
    }
    // error: should never reach here
    return tally.size()-1;
}
}

```

Jul 10, 02 13:38

StoreManager.java

Page 1/1

```
import java.util.ArrayList;

public class StoreManager
{
    /**
     * returns the name of an item whose unit price is the lowest
     * in the specified category; if no items in the specified
     * category, returns "none"
     */

    public String minUnitPrice(GroceryStore store, String category)
    {
        ArrayList list = store.getAllItems(category);
        if (list.size() == 0) {
            return "none";
        }
        GroceryItem minItem = (GroceryItem) list.get(0);

        for(int k=1; k < list.size(); k++) {
            GroceryItem current = (GroceryItem) list.get(k);
            if (current.getSize()/current.getPrice() <
                minItem.getSize()/minItem.getPrice()) {
                minItem = current;
            }
        }
        return minItem.getName();
    }
}
```