\* Assignment: RandomQueue

\*\* Student info:

- Group name: Group FRV

- Name1: Mark E. Klitgaard

- Email1: mekl@itu.dk

- Name2: Frederik T. Hoffman

- Email2: ftho@itu.dk

- Operating system: Win 7 Ultimate

- Compiler: javac

- Text editor / IDE: Notepad++

\*\* Assignment info:

Mark on of the following boxes with an X

[X] Yes, to the best of our knowledge, everything works as it

should. In particular, the main method below behaves as expected.

[ ] No, our solution does not work. (We will not get credit for this.)

Here's what doesn't work:

...

- Total hours to complete the assignment (optional): 4

Please check if true:

[X] Yes, we "avoid loitering" (in the course book's terminology) by

freeing unused references

\*\* Test client

Below is a skeleton with a test client for your solution. I suggest

you cut and paste that

//\*\*\* Class skeleton starts

import java.util.Iterator;

public class RandomQueue<Item> implements Iterable<Item>

{

// Your code goes here.

// Mine takes ca. 60 lines, by longest method has 5 lines.

public static void main(String args[])

{

// Build a queue containing the Integers 1,2,...,6:

RandomQueue<Integer> Q= new RandomQueue<Integer>();

for (int i = 1; i < 7; ++i) Q.enqueue(i); // autoboxing! cool!

// Print 30 die rolls to standard output

StdOut.print("Some die rolls: ");

for (int i = 1; i < 30; ++i) StdOut.print(Q.sample() +" ");

StdOut.println();

// Let's be more serious: do they really behave like die rolls?

int[] rolls= new int [10000];

for (int i = 0; i < 10000; ++i)

rolls[i] = Q.sample(); // autounboxing! Also cool!

StdOut.printf("Mean (should be around 3.5): %5.4f\n", StdStats.mean(rolls));

StdOut.printf("Standard deviation (should be around 1.7): %5.4f\n",

StdStats.stddev(rolls));

// Let's look at the iterator. First, we make a queue of colours:

RandomQueue<String> C= new RandomQueue<String>();

C.enqueue("red"); C.enqueue("blue"); C.enqueue("green"); C.enqueue("yellow");

Iterator I= C.iterator();

Iterator J= C.iterator();

StdOut.print("Two colours from first shuffle: ");

StdOut.print(I.next()+" ");

StdOut.print(I.next()+" ");

StdOut.print("\nEntire second shuffle: ");

while (J.hasNext()) StdOut.print(J.next()+" ");

StdOut.print("\nRemaining two colours from first shuffle: ");

StdOut.print(I.next()+" ");

StdOut.println(I.next());

}

*@SuppressWarnings("unchecked")*

*public RandomQueue() { // create an empty random queue*

*array = (Item[]) new Object[10];*

*N = 0; // number of items on the queue*

*}*

*public boolean isEmpty() { return N == 0; }*

*public int size() { return N; }*

*@SuppressWarnings("unchecked")*

*public void enqueue(Item item) { // Add item to the end of the queue.*

*boolean alreadyExists = false;*

*for(int i = 0; i<array.length; i++) {*

*if(array[i] == null) // then the end of the used array has been reached.*

*break;*

*if(item == array[i])*

*alreadyExists = true;*

*}*

*if(alreadyExists == false) {*

*if(N < array.length) {*

*array[N] = item;*

*N++;*

*}*

*else {*

*Item[] tmp = (Item[]) new Object[array.length\*2];*

*for(int i = 0; i<array.length; i++) {*

*tmp[i] = array[i];*

*}*

*array = tmp;*

*array[N] = item;*

*N++;*

*}*

*}*

*}*

*@SuppressWarnings("unchecked")*

*public Item dequeue() { // Remove a random item from the queue.*

*int p = StdRandom.uniform(N);*

*Item item = array[p];*

*if(array[p+1] != null) { //Checks if we're at the end of the used array already.*

*for(int i = p; i<array.length; i++) {*

*if(array[i+1] != null)*

*array[i] = array[i+1];*

*else //Then we've reached the end of the stored items*

*array[i] = null;*

*break;*

*}*

*}*

*else {*

*array[p] = null;*

*}*

*N--;*

*if(N == array.length/4) { //resizes the array*

*Item[] tmp = (Item[]) new Object[array.length/2];*

*for(int i = 0; i<array.length; i++) {*

*tmp[i] = array[i];*

*}*

*array = tmp;*

*}*

*return item;*

*}*

*public Item sample() {// return (but do not remove) a random item*

*int p = StdRandom.uniform(N);*

*Item item = array[p];*

*return item;*

*}*

*public Iterator<Item> iterator() { return new QueueIterator(); }*

*private class QueueIterator implements Iterator<Item> {*

*int i = 0;*

*boolean iterated[];*

*private QueueIterator() {*

*iterated = new boolean[N];*

*for(int j = 0; j<iterated.length; j++) { iterated[j] = false; }*

*}*

*public boolean hasNext() {*

*return(i<N);*

*}*

*public Item next() {*

*int j = StdRandom.uniform(N);*

*i++;*

*boolean found = false;*

*while(found == false) {*

*if(iterated[j] == false) {*

*found = true;*

*iterated[j] = true;*

*}*

*else*

*j = StdRandom.uniform(N);*

*}*

*return array[j];*

*}*

*public void remove() {}*

*}*

*}*

//\*\*\* Class skeleton ends

\*\* My output

My output looks something like this (depending of course in

interal random choices):

RandomQueue> java RandomQueue

Some die rolls: 2 1 3 2 4 6 4 6 1 1 2 3 1 3 6 1 3 1 4 3 5 5 3 2 4 2 6 3 2

Mean (should be around 3.5): 3.4975

Standard deviation (should be around 1.7): 1.6929

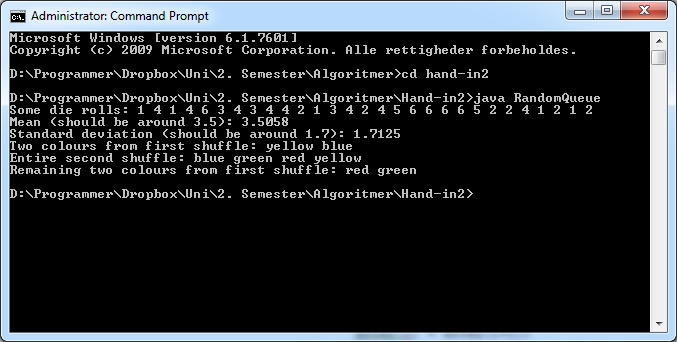
Two colours from first shuffle: blue yellow

Entire second shuffle: yellow green red blue

Remaining two colours from first shuffle: red green

\*\* Your output

Please paste the output from your code below:



\*\* Help

List whatever help (if any) that you received, including help from

TAs or fellow students. (Such help is allowed, but we want you to

acknowledge it.)

*Simon Malone, with keeping track of which items that have already been iterated. SW page 155, with help on how to make the iterator work.*

\*\* Comments

List any other comments here. Feel free to provide any feedback on

how much you learned from doing the assignment, and whether you

enjoyed doing it. In particular, tell us how this exercise could be

improved.

...