**Algorithmique / Travaux pratiques**

S11 – File d’attente





07.05.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

1. **Voici ci-dessous les méthodes du programme complétées**

// ------------------------------

// PRE: !isEmpty()

public int dequeue() {

int aux = buffer[front];

front = (front + 1) % buffer.length;

size--;

return aux;

}

// ------------------------------

private void checkSize() {

if (!(size < buffer.length)) {

int[] bufferTemp = new int[2 \* size];

int index = 0;

for (int i = front; i < buffer.length; i++) {

bufferTemp[index] = buffer[i];

index++;

}

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

bufferTemp[index] = buffer[i];

index++;

}

front = 0;

back = --index;

buffer = bufferTemp;

}

}

1. Voici ci-dessous la méthode à compléter **avec les éléments chaînés**

QueueNode chainz = new QueueNode(elt);

if (isEmpty()) {

back = chainz;

front = chainz;

} else {

back.prev = chainz;

back = chainz;

}

}

# Voici les réponses avec la méthode demo()

## Voici ci-dessous la classe QueueChained

package s11;

import java.util.Random;

public class QueueChained<E> {

// ======================================================================

static class QueueNode<E> {

E elt;

QueueNode prev = null;

// ----------

QueueNode(E elt) {

this.elt = elt;

}

}

// ======================================================================

private QueueNode front;

private QueueNode back;

// ------------------------------

public QueueChained() {

}

// --------------------------

public void enqueue(E elt) {

QueueNode<E> chainz = new QueueNode<E>(elt);

if (isEmpty()) {

back = chainz;

front = chainz;

} else {

back.prev = chainz;

back = chainz;

}

}

// --------------------------

public boolean isEmpty() {

return back == null; }

// --------------------------

// PRE : !isEmpty()

public E consult() {

return (E) front.elt;

}

// --------------------------

// PRE : !isEmpty()

public E dequeue() {

E e = (E) front.elt;

if (front == back) {

back = null;

front = null;

} else {

front = front.prev;

}

return e;

}

// --------------------------

public String toString() {

String res = "";

QueueNode<E> c = front;

while (c != null) {

res += c.elt + " ";

c = c.prev;

}

return res;

}

// ======================================================================

// ======================================================================

public static void main(String[] args) {

int n = 1000000;

if (args.length == 1)

n = Integer.parseInt(args[0]);

Random r = new Random();

long seed = r.nextInt(1000);

r.setSeed(seed);

System.out.println("Using seed " + seed);

QueueChained q = new QueueChained();

int m = 0;

int k = 0;

int p = 0;

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

boolean doAdd = r.nextBoolean();

if (doAdd) {

k++;

q.enqueue(k);

ok(!q.isEmpty(), "should be non-empty " + m + " " + k + " " + p

+ "\n");

m++;

// System.out.print("a("+k+")");

} else {

if (m == 0) {

ok(q.isEmpty(), "should be empty " + m + " " + k + " " + p

+ "\n");

} else {

ok(!q.isEmpty(), "should be non-empty " + m + " " + k + " "

+ p + "\n");

int e = (int) q.dequeue();

// System.out.print("r("+e+")");

m--;

ok(e == p + 1, "not FIFO " + m + " " + k + " " + p + "\n");

p++;

} } }

System.out.println("Test passed successfully");

demo(10);

}

static void demo(int n) {

QueueChained f;

int i, sum = 0;

f = new QueueChained();

for (i = 0; i < n; i++)

f.enqueue(i);

while (!f.isEmpty())

sum = sum + (int) f.dequeue();

System.out.println(sum);

}

// ------------------------------------------------------------

static void ok(boolean b, String s) {

if (b)

return;

throw new RuntimeException("property not verified: " + s);

}

}

## Voici ci-dessous la classe demo avec la classe générique

static void demo(int n) {

QueueChained f;

int i, sum = 0;

f = new QueueChained();

for (i = 0; i < n; i++)

f.enqueue(i);

while (!f.isEmpty())

sum = sum + (int) f.dequeue();

System.out.println(sum);

}

## Voici ci-dessous la classe ObjQueue suivi de la méthode demo()

package s11;

import java.util.Random;

public class ObjQueue {

// ======================================================================

static class QueueNode {

Object elt;

QueueNode prev = null;

// ----------

QueueNode(Object elt) {

this.elt = elt;

}

}

// ======================================================================

private QueueNode front;

private QueueNode back;

// ------------------------------

public ObjQueue() {

}

// --------------------------

public void enqueue(Object elt) {

QueueNode chainz = new QueueNode(elt);

if (isEmpty()) {

back = chainz;

front = chainz;

} else {

back.prev = chainz;

back = chainz;

}

}

// --------------------------

public boolean isEmpty() {

return back == null;

}

// --------------------------

// PRE : !isEmpty()

public Object consult() {

return front.elt;

}

// --------------------------

// PRE : !isEmpty()

public Object dequeue() {

Object e = front.elt;

if (front == back) {

back = null;

front = null;

} else {

front = front.prev;

}

return e;

}

// --------------------------

public String toString() {

String res = "";

QueueNode c = front;

while (c != null) {

res += c.elt + " ";

c = c.prev;

}

return res;

}

// ======================================================================

// ======================================================================

public static void main(String[] args) {

int n = 1000000;

if (args.length == 1)

n = Integer.parseInt(args[0]);

Random r = new Random();

long seed = r.nextInt(1000);

r.setSeed(seed);

System.out.println("Using seed " + seed);

ObjQueue q = new ObjQueue();

int m = 0;

int k = 0;

int p = 0;

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

boolean doAdd = r.nextBoolean();

if (doAdd) {

k++;

q.enqueue(k);

ok(!q.isEmpty(), "should be non-empty " + m + " " + k + " " + p

+ "\n");

m++;

// System.out.print("a("+k+")");

} else {

if (m == 0) {

ok(q.isEmpty(), "should be empty " + m + " " + k + " " + p

+ "\n");

} else {

ok(!q.isEmpty(), "should be non-empty " + m + " " + k + " "

+ p + "\n");

int e = (int) q.dequeue();

// System.out.print("r("+e+")");

m--;

ok(e == p + 1, "not FIFO " + m + " " + k + " " + p + "\n");

p++;

}

}

}

System.out.println("Test passed successfully"); }

// ------------------------------------------------------------

static void ok(boolean b, String s) {

if (b)

return;

throw new RuntimeException("property not verified: " + s); }

}

Méthode demo():

static void demo(int n) {

ObjQueue f;

int i, sum = 0;

f = new ObjQueue();

for (i = 0; i < n; i++)

f.enqueue(new Integer(i));

while (!f.isEmpty())

sum = sum + (Integer) f.dequeue();

System.out.println(sum);

}