Queue dan Priority Queue

Arna F

Queue

- Queue (antrian) adalah penyimpanan item yang dapat diakses melalui front dan back dari antrian. Item masuk pada back dan keluar dari front

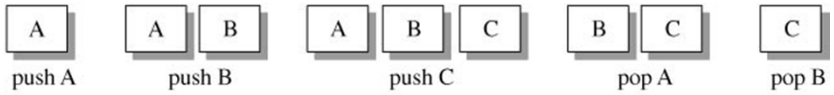
- Contoh queue misalnya antrian pada supermarket atau bank

Operasi pada Queue

- Operasi pada queue terjadi pada posisi front dan back

  push(item) menambah item pada back
  pop() menghapus elemen dari front
  peek() mengakses nilai pada front

- Item yang dihapus (pop) dariqueue adalah elemen pertama yang ditambah (push) ke queue. Queue merupakan penyimpanan secara FIFO (first-in-first-out)



Interface Queue

- Interface Queue generic mendefinisikan operasi yang mengakses dan meng-update elemen hanya pada akhir list

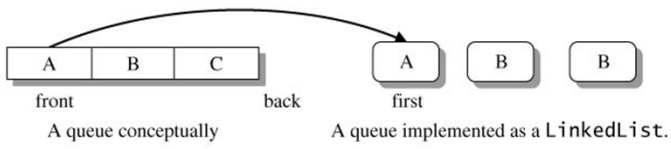
interface Queue<T>	**ds.util**
boolean	**isEmpty**() menghasilkan nilai true jika tidak ada elemen dan false jika sedikitnya satu elemen
T	**peek**() menghasilkan elemen pada awal queue. Jika kosong, throws NoSuchElementException.

Interface Queue (lanj.)

interface Queue<T>	ds.util
T	**pop**() menghapus elemen di awal queue dan menghasilkan nilai. Jika queue kosong, throws NoSuchElementException.
void	**push**(T item) menambah item pada posisi akhir queue.
int	**size**() menghasilkan jumlah elemen pada queue

Class LinkedQueue

- Interface Queue mendefinisikan method yang terbatas. Class queue digunakan untuk mengimplementasikan queue menggunakan struktur linked list. Class LinkedQueue menggunakan collection LinkedList



Class LinkedQueue (lanj)

```
public class LinkedQueue<T> implements Queue<T>
{
   private LinkedList<T> qlist = null;
   public LinkedQueue ()
   {
      qlist = new LinkedList<T>();
   }
   . . .
}
```

Implementasi LinkedQueue Method pop()

- Method mempunyai efisiensi runtime O(1)

```
public T pop()
{
   // if the queue is empty, throw
   // NoSuchElementException
   if (isEmpty())
      throw new NoSuchElementException(
            "LinkedQueue pop(): queue empty");

   // remove and return the first element in the list
   return qlist.removeFirst();
}
```

Program 15.1

- Program berikut mengimplementasikan penjadwalan interview yang berupa antrian obyek Time24. Output berupa waktu dan panjang setiap interview.

Program 15.1 (lanj)

```
import java.io.*;
import java.util.Scanner;
import ds.util.LinkedQueue;
import ds.time.Time24;

public class Program15_1
{
   public static void main(String[] args)
   throws IOException
   {
      final Time24 END_DAY = new Time24(17,00);
      String apptStr;

      // time interval from current appt to next appt
      Time24 apptTime = null, interviewTime = null;

      // input stream to read times as strings from
      // file "appt.dat"
      Scanner input = new Scanner(
            new FileReader("appt.dat"));
```

Program 15.1 (lanj)

```
// queue to hold appointment time
// for job applicants
LinkedQueue<Time24> apptQ = new
      LinkedQueue<Time24>();

// construct the queue by appt times as
// strings from file; use parseTime to
// convert to Time24 object
while (input.hasNext())
{
   apptStr = input.nextLine();
   apptQ.push(Time24.parseTime(apptStr));
}

// output the day's appointment schedule
System.out.println("Appointment   Interview");
```

Program 15.1 (continued)

```
// pop next appt time and determine
// available time for interview (peek
// at next appt at front of queue)
while (!apptQ.isEmpty())
{
   // get the next appointment
   apptTime = apptQ.pop();

   // interview time is interval to next
   // appt or to END_DAY
   if (!apptQ.isEmpty())
      interviewTime = apptTime.interval(
            apptQ.peek());
   else
      interviewTime = apptTime.interval(END_DAY);
   // display appointment time and interview time
   System.out.println("   " + apptTime +
         "        " + interviewTime);
}
   }
}
```

Program 15.1 (Run)

```
File "appt.dat":

10:00
11:15
13:00
13:45
14:30
15:30
16:30

Run:

Appointment    Interview
   10:00          1:15
   11:15          1:45
   13:00          0:45
   13:45          0:45
   14:30          1:00
   15:30          1:00
   16:30          0:30
```

Bounded Queue

- Bounded queue adalah queue yang berisi elemen terbatas. Menambah queue terjadi hanya jika queue tidak penuh.
- Class BQueue mengimplementasikan bounded queue. Class mengimplementasikan interface Queue interface. Method boolean full() menandakan queue penuh.
- Class menggunakan array untuk menyimpan elemen.

Class API BQueue

interface BQueue<T> implements Queue	ds.util
	BQueue() Membuat queue dengan ukuran 50.
	BQueue(int size) Membuat queue dengan ukiuran size.
boolean	**full**() menghasilkan true jika jumlah element dalam queue sama dengan ukuran size dan sebaliknya bernilai false.

Contoh Class BQueue

- Contoh berikut mengilustrasikan deklarasi obyek BQueue dan menggunakan full() untuk mencegah penambahan ke queue yang penuh. Exception terjadi jika memanggil push() dalam try block dan menambah elemen ke queue yang penuh.

Contoh Class BQueue (lanj)

```
// declare an empty bounded queue with fixed size 15
BQueue<Integer> q = new BQueue<Integer>(15);
int i;

// fill-up the queue
for (i=1; !q.full(); i++)
   q.push(i);

// output element at the front of q and the queue size
System.out.println(q.peek() + " " + q.size());

try
{
   q.push(40); // exception occurs
}

catch (IndexOutOfBoundsException iobe)
{  System.out.println(iobe); }
```

Contoh Class BQueue (hasil)

```
Output:
   1 15
   java.lang.IndexOutOfBoundsException: BQueue push(): queue full
```

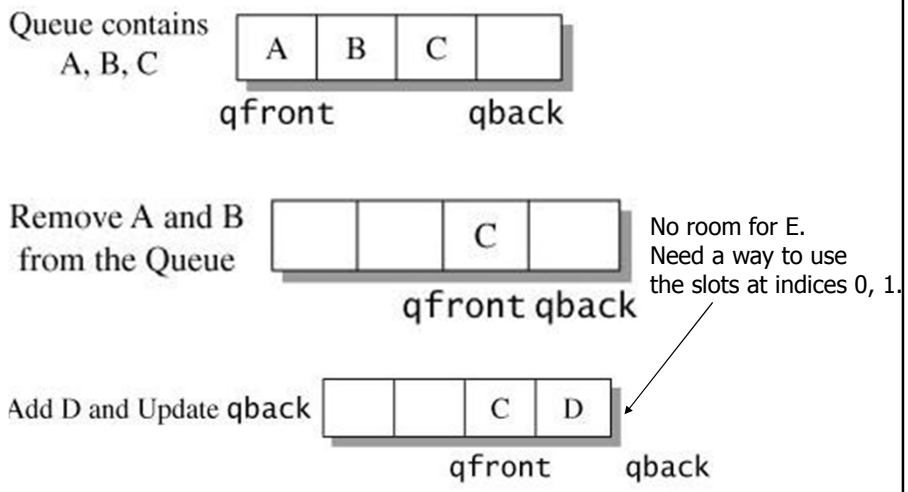
Class BQueue

```
public class BQueue<T> implements Queue<T>
{
   // array holding the queue elements
   private T[] queueArray;
   // index of the front and back of the queue
   private int qfront, qback;
   // the capacity of the queue and the current size
   private int qcapacity, qcount;

   // create an empty bounded queue with specified size
   public BQueue(int size)
   {
      qcapacity = size;
      queueArray = (T[])new Object[qcapacity];
      qfront = 0;
      qback = 0;
      qcount = 0;
   }
```

Class Bqueue (hasil)

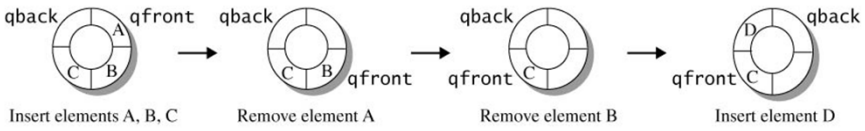
```
   public BQueue()
   {
      // called non-default constructor
      // with capacity = 50
      BQueue(50);
   }

   < method full() and methods in the Queue interface >
}
```

Implementasi Class BQueue



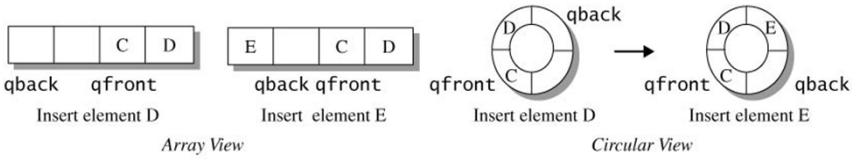
Implementasi Class BQueue (lanj)

- Asumsikan queue sebagai circular sequence dengan sekumpulan slot dimana elemen masuk searah jarum jam. Elemen pada index qfront keluar dari queue dan elemen masuk ke queue pada index qback.



Implementasi Class BQueue (lanj)

- Memberlakukan array sebagai circular sequence menyebabkan qfront dan qback berubah dari back ke front dari array jika melewati akhir array.




```
Move qback forward: qback = (qback + 1) % qcapacity;
Move qfront forward:qfront = (qfront + 1) % qcapacity;
```

Class BQueue full()

```
public boolean full()
{
   return qcount == qcapacity;
}
```

Class BQueue push()

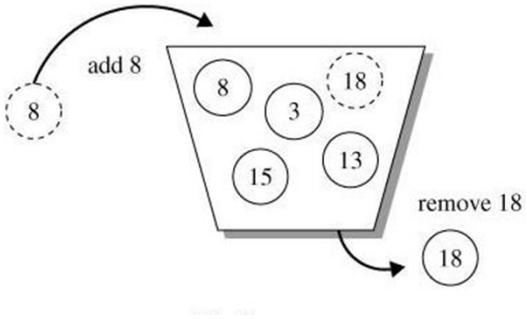
```
public void push(T item)
{
   // is queue full? if so, throw an
   // IndexOutOfBoundsException
   if (qcount == qcapacity)
      throw new IndexOutOfBoundsException(
            "BQueue push(): queue full");

   // insert into the circular queue
   queueArray[qback] = item;
   qback = (qback+1) % qcapacity;

   // increment the queue size
   qcount++;
}
```

Class BQueue pop()

```
public T pop()
{
   // if queue is empty, throw a
   // NoSuchElementException
   if (count == 0)
      throw new NoSuchElementException(
            "BQueue pop(): empty queue");

   // save the front of the queue
   T queueFront = queueArray[qfront];

   // perform a circular queue deletion
   qfront = (qfront+1) % qcapacity;

   // decrement the queue size
   qcount--;

   // return the front
   return queueFront;
}
```

Priority Queue Collection

- Priority queue adalah kumpulan semua elemen yang mempunyai urutan perbandingan (priority).
- Menyediakan akses sederhana dan operasi update dimana penghapusan selalu menghapus elemen dengan prioritas tertinggi.



Interface PQueue

- PQueue merupakan interface generic yang membuat queue dengan nama yang sama.

interface PQueue<T>	**ds.util**
boolean	**isEmpty**() bernilai true jika priority queue kosong dan sebagainya false.
T	**peek**() menghasilkan nilai item dengan prioritas tertinggi. Jika kosong, throw NoSuchElementException.

Interface Pqueue (lanj)

interface PQueue<T>	ds.util
T	**pop**() menghasilkan prioritas tertinggi dari queue dan menghasilkan nilai. Jika kosong, throw NoSuchElementException.
void	**push**(T item) menyisipkan item ke queue prioritas.
int	**size**() menghasilkan jumlah elemen dalam priority queue

Class HeapPQueue

- Class HeapPQueue mengimplementasikan interface PQueue.
  - Secara default, elemen dari prioritas tertinggi adalah yang mempunyai nilai tertinggi (maksimum priority queue); artinya, jika x dan y adalah dua element dalam priority queue dan x > y, maka x mempunyai prioritas tertinggi daripada y.

Contoh Class HeapPQueue

```
// create an empty priority queue of generic type String
HeapPQueue<String> pq = new HeapPQueue<String>();
int n;
pq.push("green");
pq.push("red");
pq.push("blue");
// output the size and element with the highest priority
System.out.println(pq.size() + "  " + pq.peek());
// use pop() to clear the collection and list elements in
// priority (descending) order
while (!pq.isEmpty())
       System.out.print(pq.pop() + "  ");
```

```
Output:
   3  red
   red  green  blue
```

Support Services Pool

- Aplikasi ini memproses permintaan pekerjaan ke company support service pool. Sebuah request mempunyai job ID, job status, dan time requirement.
- JobStatus adalah tipe enum yang berisi daftar kategori karyawan yang nilainya digunakan untuk membandingkan obyek.
- Class JobRequest mengimplementasikan Comparable dan menggambarkan obyek job.

```
enum JobStatus
{
        clerk (0), manager (1), director(2),
     president(3);
        int jsValue;
        JobStatus(int value) { jsValue = value; }
        public int value() { return jsValue; }
}
```

Support Services Pool (lanj)

class JOBREQUEST implements Comparable<JobRequest>	
	Constructors
	JobRequest (JobStatus status, int ID, int time) membuat obyek dengan argumen tertentu.
	Methods
int	**getJobID**() menghasilkan nilai ID obyek.
int	**getJobStatus**() menghasilkan status obyek.

Support Services Pool (lanj.)

class JOBREQUEST implements Comparable<JobRequest>	
int	**getJobTime**() menghasilkan waktu obyek dalam menit.
static JobRequest	**readJob**(Scanner sc) membaca job dari scanner dalam bentuk status jobID jobTime; menghasilkan obyek JobRequest atau null dari input file sampai akhir file.
String	**toString**() menghasilkan string yang merepresentasikan job dalam format "<status name> <ID> <time>".
int	**compareTo**(JobRequest item) membandingkan obyek jobStatus saat ini dengan jobStatus dari setiap item.

Program 15.3

- Program memproses job request dengan status karyawan yang berbeda. Output menampilkan daftar status job dengan total waktu.

Program 15.3 (lanj)

```
import java.io.*;
import java.util.Scanner;
import ds.util.HeapPQueue;

public class Program15_3
{
   public static void main(String[] args)
   throws IOException
   {
      // handle job requests
      HeapPQueue<JobRequest> jobPool =
            new HeapPQueue<JobRequest>();

      // job requests are read from file "job.dat"
      Scanner sc = new Scanner(new FileReader(
            "job.dat"));
```

Program 15.3 (lanj)

```
// time spent working for each category
// of employee
// initial time 0 for each category
int[] jobServicesUse = {0,0,0,0};
JobRequest job = null;

// read file; insert each job into
// priority queue
while ((job = JobRequest.readJob(sc)) != null)
   jobPool.push(job);

// delete jobs from priority queue
// and output information
System.out.println("Category     Job ID" +
      "    Job Time");
```

Program 15.3 (lanj)

```
while (!jobPool.isEmpty())
{
   // remove a job from the priority
   // queue and output it
   job = (JobRequest)jobPool.pop();
   System.out.println(job);

   // accumulate job time for the
   // category of employee
   jobServicesUse[job.getStatus().value()] +=
         job.getJobTime();
}
System.out.println();

   writeJobSummary(jobServicesUse);
}
```

Program 15.3 (hasil)

```
    private static void writeJobSummary(
    int[] jobServicesUse)
    {
       System.out.println("Total Pool Usage");
       System.out.println("   President    " +
             jobServicesUse[3]);
       System.out.println("   Director     " +
             jobServicesUse[2]);
       System.out.println("   Manager      " +
             jobServicesUse[1]);
       System.out.println("   Clerk        " +
             jobServicesUse[0]);
    }
 }
```

Program 15.3 (Run)

```
Run

Category      Job ID     Job Time
President       303         25
President       306         50
Director        300         20
Director        307         70
Director        310         60
Director        302         40
Manager         311         30
Manager         304         10
Manager         305         40
Clerk           308         20
Clerk           309         20
Clerk           301         30

Total Pool Usage
   President     75
   Director      190
   Manager       80
   Clerk         70
```