加入队列。第 16 行让一个病人移出队列。

程序清单 24-10 TestPriorityQueue.java

```
1  public class TestPriorityQueue {
2    public static void main(String[] args) {
3      Patient patient1 = new Patient("John", 2);
4      Patient patient2 = new Patient("Jim", 1);
5      Patient patient3 = new Patient("Tim", 5);
6      Patient patient4 = new Patient("Cindy", 7);
7
8      MyPriorityQueue<Patient> priorityQueue
9        = new MyPriorityQueue<>();
10     priorityQueue.enqueue(patient1);
11     priorityQueue.enqueue(patient2);
12     priorityQueue.enqueue(patient3);
13     priorityQueue.enqueue(patient4);
14
15     while (priorityQueue.getSize() > 0)
16       System.out.print(priorityQueue.dequeue() + " ");
17   }
18
19   static class Patient implements Comparable<Patient> {
20     private String name;
21     private int priority;
22
23     public Patient(String name, int priority) {
24       this.name = name;
25       this.priority = priority;
26     }
27
28     @Override
29     public String toString() {
30       return name + "(priority:" + priority + ")";
31     }
32
33     @Override
34     public int compareTo(Patient patient) {
35       return this.priority - patient.priority;
36     }
37   }
38 }
```

```
Cindy(priority:7) Tim(priority:5) John(priority:2) Jim(priority:1)
```

复习题

24.21 什么是优先队列?

24.22 MyProrityQueue 中的 enqueue、dequeue 以及 getSize 方法的时间复杂度为多少?

24.23 下面语句哪些有错误?

```
1  MyPriorityQueue<Object> q1 = new MyPriorityQueue<>();
2  MyPriorityQueue<Number> q2 = new MyPriorityQueue<>();
3  MyPriorityQueue<Integer> q3 = new MyPriorityQueue<>();
4  MyPriorityQueue<Date> q4 = new MyPriorityQueue<>();
5  MyPriorityQueue<String> q5 = new MyPriorityQueue<>();
```

本章小结

1. 本章学习了如何实现数组线性表、链表、栈以及队列。
2. 定义一个数据结构本质上是定义一个类。为数据结构定义的类应该使用数据域来存储数据，并提供方法来支持诸如插入和删除等操作。

3. 创建一个数据结构是从该类创建一个实例。这样就可以将方法应用在实例上来处理数据结构，比如插入一个元素到数据结构中，或者从数据结构中删除一个元素。
4. 本章学习了如何采用堆来实现一个优先队列。

测试题

回答位于网址 www.cs.armstrong.edu/liang/intro10e/test.html 的本章测试题。

编程练习题

24.1 （在 MyList 中添加集合操作）在 MyList 中定义下列方法，并在 MyAbstractList 中实现下列方法：

```
/** Adds the elements in otherList to this list.
  * Returns true if this list changed as a result of the call */
public boolean addAll(MyList<E> otherList);

/** Removes all the elements in otherList from this list
  * Returns true if this list changed as a result of the call */
public boolean removeAll(MyList<E> otherList);

/** Retains the elements in this list that are also in otherList
  * Returns true if this list changed as a result of the call */
public boolean retainAll(MyList<E> otherList);
```

编写一个测试程序，创建两个 MyArrayList 对象 list1 和 list2，初始值分别为 {"Tom", "George","Peter","Jean","Jane"} 和 {"Tom","George","Michael","Michelle", "Daniel"}。然后，执行以下操作：

- 调用方法 list1.addAll(list2)，并显示 list1 和 list2。
- 采用同样的初始值重新创建 list1 和 list2，然后调用 list1.removeAll(list2)，并显示 list1 和 list2。
- 采用同样的初始值重新创建 list1 和 list2，然后调用 list1.retainAll(list2)，并显示 list1 和 list2。

*24.2 （实现 MyLinkedList）本书中省略了下述方法的实现，请实现它们：contains(E e)、get(int index)、indexOf(E e)、lastIndexOf(E e) 和 set(int index, E e)。

*24.3 （实现双向链表）程序清单 24-6 中使用的 MyLinkedList 类创建了一个单向链表，它只能单向遍历线性表。修改 Node 类，添加一个名为 previous 的数据域，让它指向链表中的前一个结点，如下所示：

```
public class Node<E> {
  E element;
  Node<E> next;
  Node<E> previous;

  public Node(E e) {
    element = e;
  }
}
```

实现一个名为 TwoWayLinkedList 的新类，使用双向链表来存储元素。课本中的 MyLinkedList 类继承自 MyAbstractList。定义 TwoWayLinkedList 继承 java.util.AbstractSequentialList 类。不光要实现 listIterator() 和 listIterator(int index) 方法，还要实现定义在 MyLinkedList 中包括的所有方法。都返回一个 java.util.ListIterator<E> 类型的实例。前者指向线性表的头部，后者指向指定下标的元素。

24.4 （使用 GenericStack 类）编写一个程序，以降序显示前 50 个素数。使用栈存储素数。

24.5 （利用继承关系实现 GenericQueue）24.5 节使用组合关系实现了 GenericQueue。继承 java.util.LinkedList 类，创建一个新的队列类。

*24.6 （使用 Comparator 实现泛型 PriorityQueue）修改程序清单 24-9 中的 MyPriorityQueue，使用一个泛型参数来比较对象。如下定义一个使用 Comparator 作为参数的新的构造方法：

```
PriorityQueue(Comparator<? super E> comparator)
```

**24.7 （动画：链表）编写一个程序，用动画实现链表的查找、插入和删除，如图 24-1b 所示。按钮 Search 用来查找一个指定的值是否在链表中；按钮 Delete 用来从链表中删除一个特定值；按钮 Insert 用来在链表的特定下标处插入一个值，如果没有指定下标，则添加到链表的末尾。

*24.8 （动画：数组线性表）编写一个程序，用动画实现数组线性表的查找、插入和删除，如图 24-1a 所示。按钮 Search 用来查找一个指定的值是否在线性表中；按钮 Delete 用来从线性表中删除一个特定值；按钮 Insert 用来在链表的特定下标处插入一个值，如果没有指定下标，则添加到链表的末尾。

*24.9 （动画：慢动作显示数组线性表）改进前面编程练习题，通过慢动作显示插入和删除操作，如网址 http://www.cs.armstrong.edu/liang/animation/ArrayListAnimationInSlowMotion.html 所示。

*24.10 （动画：栈）编写一个程序，用动画实现栈的压入和弹出，如图 24-20a 所示。

*24.11 （动画：双向链表）编写一个程序，用动画实现双向链表的查找、插入和删除，如图 24-24 所示。按钮 Search 用来查找一个指定的值是否在线性表中；按钮 Delete 用来从线性表中删除一个特定值；按钮 Insert 用来在链表的特定下标处插入一个值，如果没有指定下标，则添加到链表的末尾。同时，添加两个名为 Forward Traversal 和 Backward Traversal 的按钮，用于采用遍历器分别以向前和往后的方式来显示元素。

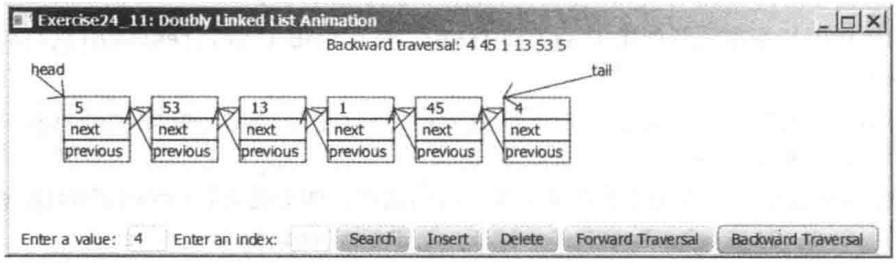

图 24-24 程序实现双向链表的运行动画

*24.12 （动画：队列）编写一个程序，用动画实现队列的 enqueue 和 dequeue 操作，如图 24-20b 所示。

*24.13 （斐波那契数遍历器）定义一个名为 FibonacciIterator 的遍历器，用于遍历斐波那契数字。构造方法带有一个参数，用于指定斐波那契数字的上限。比如，new FibonacciIterator (23302) 创建一个遍历器，可以用于遍历小于或者等于 23302 的斐波那契数。编写一个测试程序，使用该遍历器显示所有小于或者等于 100000 的斐波那契数。

*24.14 （素数遍历器）定义一个名为 PrimeIterator 的遍历器，用于遍历素数。构造方法带有一个参数，用于指定斐波那契数字的上限。比如，new PrimeIterator (23302) 创建一个遍历器，可以用于遍历小于或者等于 23302 的素数。编写一个测试程序，使用该遍历器显示所有小于或者等于 100000 的素数。

**24.15 （测试 MyArrayList）设计和编写一个完整的测试程序，用于测试程序清单 24-3 中的 MyArrayList 类是否符合所有的要求。

**24.16 （测试 MyLinkedList）设计和编写一个完整的测试程序，用于测试程序清单 24-6 中的 MyLinkedList 类是否符合所有的要求。