**package** queues;

**public** **interface** IQueues {

/\*\* Array-based queue implementation \*/

**class** AQueue<E> **implements** Queue<E> {

**private** **static** **final** **int** ***defaultSize*** = 10;

**private** **int** maxSize;

**private** **int** front;

**private** **int** rear;

**private** E[] listArray;

// Maximum size of queue

// Index of front element

// Index of rear element

// Array holding queue elements

/\*\* Constructors \*/

AQueue() { **this**(***defaultSize***); }

@SuppressWarnings("unchecked") // For generic array

AQueue(**int** size) {

maxSize = size+1; // One extra space is allocated

rear = 0; front = 1;

listArray = (E[])**new** Object[maxSize]; // Create listArray

}

/\*\* Reinitialize \*/

**public** **void** clear()

{ rear = 0; front = 1; }

/\*\* Put "it" in queue \*/

**public** **void** enqueue(E it) {

**assert** ((rear+2) % maxSize) != front : "Queue is full";

rear = (rear+1) % maxSize; // Circular increment

listArray[rear] = it;

}

/\*\* Remove and return front value \*/

**public** E dequeue() {

**assert** length() != 0 : "Queue is empty";

E it = listArray[front];

front = (front+1) % maxSize; // Circular increment

**return** it;

}

/\*\* **@return** Front value \*/

**public** E frontValue() {

**assert** length() != 0 : "Queue is empty";

**return** listArray[front];

}

/\*\* **@return** Queue size \*/

**public** **int** length()

{ **return** ((rear+maxSize) - front + 1) % maxSize; }

Figure 4.27 An array-based queue implementation.

130 Chap. 4 Lists, Stacks, and Queues

/\*\* Linked queue implementation \*/

**class** LQueue<E> **implements** Queue<E> {

**private** Link<E> front;

**private** Link<E> rear;

**private** **int** size;

// Pointer to front queue node

// Pointer to rear queuenode

// Number of elements in queue

/\*\* Constructors \*/

**public** LQueue() { init(); }

**public** LQueue(**int** size) { init(); } // Ignore size

/\*\* Initialize queue \*/

**private** **void** init() {

front = rear = **new** Link<E>(**null**);

size = 0; }

/\*\* Reinitialize queue \*/

**public** **void** clear() { init(); }

/\*\* Put element on rear \*/

**public** **void** enqueue(E it) {

rear.setNext(**new** Link<E>(it, **null**));

rear = rear.next();

size++;

}

/\*\* Remove and return element from front \*/

**public** E dequeue() {

**assert** size != 0 : "Queue is empty";

E it = front.next().element(); // Store dequeued value

front.setNext(front.next().next()); // Advance front

**if** (front.next() == **null**) rear = front; // Last Object

size--;

**return** it; // Return Object

}

/\*\* **@return** Front element \*/

**public** E frontValue() {

**assert** size != 0 : "Queue is empty";

**return** front.next().element();

}

/\*\* **@return** Queue size \*/

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

}