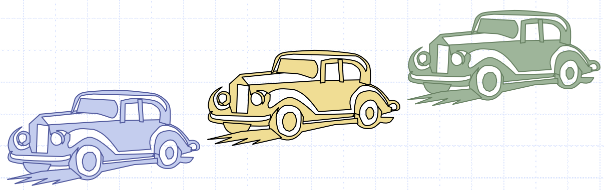
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Queues



The Queue ADT

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(object): inserts an element at the end of the queue
  - object dequeue(): removes and returns the element at the front of the queue
- Auxiliary queue operations:
  - object first(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored
- Boundary cases:
  - Attempting the execution of dequeue or first on an empty queue returns null

Example

Operation		*Output* *Q*
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
first()	*7*	(7)
dequeue()	*7*	()
dequeue()	*null*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

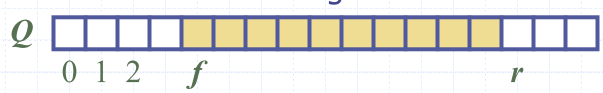
Applications of Queues

- Direct applications
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

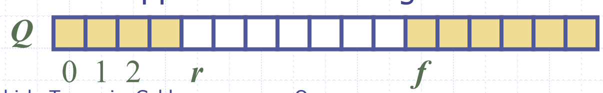
Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and size
  - f index of the front element
  - sz number of stored elements
- When the queue has fewer than N elements, array location $r = (f + sz)\ mod\ N$ is the first empty slot past the rear of the queue

normal configuration



wrapped-around configuration

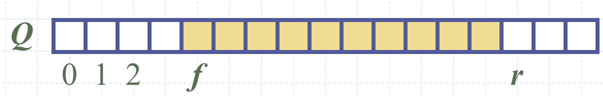


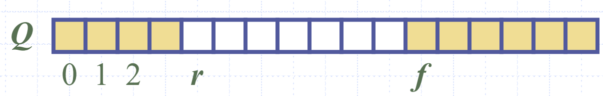
Queue Operations

- We use the modulo operator (remainder of division)

Algorithm *size*()
  return *sz*

Algorithm *isEmpty*()
  return (*sz* == 0)

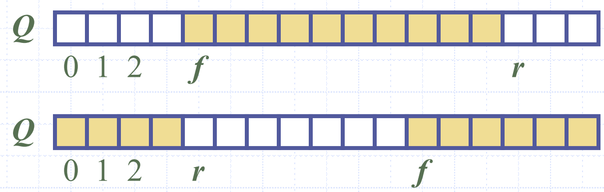




Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

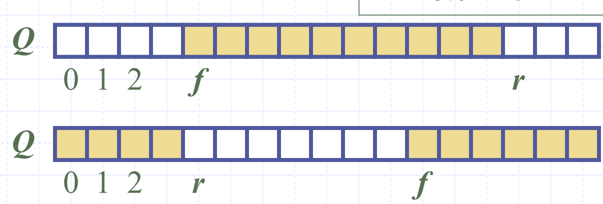
Algorithm *enqueue*(o)
  if *size*() $= N - 1$ **then**
    throw *IllegalStateException*
  else
    $r \leftarrow (f + sz) \bmod N$
    $Q[r] \leftarrow o$
    $sz \leftarrow (sz + 1)$



Queue Operations (cont.)

- Note that operation dequeue returns null if the queue is empty

Algorithm *dequeue*()
  if *isEmpty*() **then**
    return *null*
  else
    $o \leftarrow Q[f]$
    $f \leftarrow (f + 1) \bmod N$
    $sz \leftarrow (sz - 1)$
    return o



Queue Interface in Java

- Java interface corresponding to our Queue ADT
- Assumes that first() and dequeue() return null if queue is empty

```
public interface Queue<E> {
  int size();
  boolean isEmpty();
  E first();
  void enqueue(E e);
  E dequeue();
}
```


Array-based Implementation

```
/** Implementation of the queue ADT using a fixed-length array. */
public class ArrayQueue<E> implements Queue<E> {
  // instance variables
  private E[ ] data;                          // generic array used for storage
  private int f = 0;                          // index of the front element
  private int sz = 0;                         // current number of elements

  // constructors
  public ArrayQueue( ) {this(CAPACITY);}      // constructs queue with default capacity
  public ArrayQueue(int capacity) {           // constructs queue with given capacity
    data = (E[ ]) new Object[capacity];       // safe cast; compiler may give warning
  }

  // methods
  /** Returns the number of elements in the queue. */
  public int size( ) { return sz; }

  /** Tests whether the queue is empty. */
  public boolean isEmpty( ) { return (sz == 0); }

```

Array-based Implementation (2)

```java
/** Inserts an element at the rear of the queue. */
public void enqueue(E e) throws IllegalStateException {
  if (sz == data.length) throw new IllegalStateException("Queue is full");
  int avail = (f + sz) % data.length;     // use modular arithmetic
  data[avail] = e;
  sz++;
}

/** Returns, but does not remove, the first element of the queue (null if empty). */
public E first() {
  if (isEmpty()) return null;
  return data[f];
}

/** Removes and returns the first element of the queue (null if empty). */
public E dequeue() {
  if (isEmpty()) return null;
  E answer = data[f];
  data[f] = null;                          // dereference to help garbage collection
  f = (f + 1) % data.length;
  sz--;
  return answer;
}
```


Comparison to java.util.Queue

- Our Queue methods and corresponding methods of java.util.Queue:

Our Queue ADT	Interface java.util.Queue	
	throws exceptions	returns special value
enqueue(e)	add(e)	offer(e)
dequeue()	remove()	poll()
first()	element()	peek()
size()	size()	
isEmpty()	isEmpty()	

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  1. e = Q.dequeue()
  2. Service element e
  3. Q.enqueue(e)