"UML" for a second **queue** class (revised 2-3-05)

NOW: for a FIXED CAPACITY queue

adapted from Ch. 8, Savitch and Main, "Data Structures and Other Objects Using C++"

Template Class: `queue`
/* a collection of items such that entries can be inserted at one end (called the **rear**) and removed at the other end (called the **front**). */

Member data and related details:
/* contains elements of the type set to be the value of template parameter **Item** */

/* has a fixed capacity */

Constructors:
/* postcondition: creates an empty **queue** instance */
```
queue( );
```

Accessors and other constant member functions:
/* postcondition: returns **true** if queue is empty, and returns **false** otherwise */
```
bool     is_empty( ) const;
```

/* postcondition: returns **true** if queue is full (if it contains the number of items equal to its capacity), and returns **false** otherwise */
```
bool     is_full( ) const;
```

/* precondition: is_empty() == false */
/* postcondition: returns the value of the front item of the queue, BUT the queue is unchanged. */
```
Item     get_front( ) const;
```

/* postcondition: returns the capacity of the queue (how many items it CAN hold) */
```
int      get_capacity( ) const;
```

/* postcondition: returns the number of elements currently in the queue */
```
int      get_size( ) const;
```

Modifiers and other modifying member functions:
/* precondition: is_full() == false */
/* postcondition: a new copy of **entry** has been inserted at the **rear** of the queue */
```
void     enqueue(const Item& entry);
```

/* precondition: is_empty() == false */
/* postcondition: the front item of the queue has been removed, and its value is returned */
```
Item     dequeue( );
```