Queues

- Object visibility: public, private, protected
- Queues - another useful ADT
  - Class interface
  - Polymorphism
  - Using a List to represent a Queue
  - Using stacks to represent a Queue
Object Visibility

- Modifiers - applied to object declarations
  - *public*: visible wherever its class is visible
    - Few instance variable examples because this breaks ADT control over access
  - *private*: visible only within its own class
    - in Binary_Expr: operand1, operand2
  - *protected*: visible to subclasses and to other classes in same package
Object Visibility

– *final*: object may not be changed

– *static*: object is a class object
  – only one exists in its class, accessed using `class_name.object_name`
  – constants as in `static final int limit = 30;`

– *no modifier*: object is visible within its package only
Queues

- Example: waiting in line for anything
- Intuitively, something resembling a stream to which you can add or from which you can remove data
- Data is always removed from the front of a queue and added to the back of a queue
- Allows data items to be removed in the order in which they entered.
- FIFO discipline: First In First Out
Queues

- Used for simulations where order is important
  - e.g. restaurant patrons of one waiter, requests to use the printer received by an operating system, requests for special permission numbers by students
- Implementation is hidden from user and can be changed without changing program behavior
What are the primitive operations of a queue?
How to hold objects in a queue?
How to provide access to them?
How to represent an empty queue?
Must design class to avoid special cases and to be efficient. How?
  use an array?
  use a linked list?
Queue Class: Instance Methods

- void enter(Object newItem)
- Object remove () throws QueueException
- Object peek() throws QueueException
- int getLength()
- boolean empty()
- String toString()
- Enumeration getEnumeration()
Instance Methods Specification

- void enter(Object newItem)
  - adds new element to end of (back of) queue
- int getLength()
  - returns current number of items in queue
- boolean empty()
  - returns true if queue is empty, else false
Instance Methods Specification

• String toString()
  – usual conversion for printing a queue object
  – uses individual toString() methods on each element

• Enumeration getEnumeration()
  – returns enumeration object corresponding to Queue receiver
  – Queue object not altered by enumeration
Instance Method Specification

• Object remove() throws QueueException
  – removes an element from front of queue
• Object peek() throws QueueException
  – returns the element at the front of queue WITHOUT removing it!
• remove() and peek() throw QueueException when invoked on an empty queue
• similar to Stack’s pop() and peek()
Queue Representation

• How to avoid special cases?
  – Adding to an empty queue
  – Removing from a queue with only 1 element

• Idea: use a fake header in front of every queue
  – Only subList field will contain significant information
  – Then special cases are eliminated (How?)
Queue Representation

- **Front** - a list
  - subList field contains actual list with info
  - info field not used
- **Back** - a list
  - info field contains last object in queue
  - subList field is null
  - is “innermost” list in queue representation
Queue Representation

empty queue:

non-empty queue:

? is dummy info field
public void enter (Object newItem) {
    List nl = new List (newItem, null);
    List oldBack = back;
    oldBack.subList = nl;
    back = nl;
    length++;
}

receiver:    newItem:  E
Enter - How it works?

this:

nl:

newItem: E

front

back

front

back

oldBack
Enter - How it works?

front              back
oldBack

front              back
oldBack
Enter( E ) on Empty Queue

1. ? \( \lambda \)  
   front

   nl  
   back

    E \( \lambda \)

2. ? \( \lambda \)  
   front

   Oldback  
   back

   ? \( \lambda \)  
   back
Enter(E) on empty Queue

3. Note: this is treated just like an add to an already existing queue.
public Object remove() throws QueueException{
    if (empty())
        throw new QueueException("Attempt to remove" +
            + " from an empty queue");
    front = front.subList; //destructive operation
    length--; //changes receiver queue
    return front.info;
}
**remove() on empty Queue**

Note: length is 0

whereas null.empty() yields NullPointerException
Method peek()

public Object peek() throws QueueException{
    if (empty())
        throw new QueueException(" Attempt to peek"+
        " at an empty queue");
    return front.subList.info;
}//note no decrement to length here

remains as

and F is returned
**Method toString() in Queue**

```java
public String toString(){
    String ret = "Queue length is " +
                Integer.toString(getLength()) + "\n";
    Enumeration qe = getEnumeration();
    String line = "";//empty String
    while (qe.hasMoreElements()) {
        line = line + (qe.nextElement()).toString();
        if (line.length() > 60){
            ret = ret + line + "\n";
            line = " ";
        }
    else line = line + " ";
    }
    return (ret + line + "\n");
}
```

**polymorphism**

choice of which toString() to call based on run-time type of object extracted