IT 179
13
Queues

Queue

- The queue, like the stack, is a widely used data structure
- A queue differs from a stack in one important way
  - A stack is LIFO list, *Last-In, First-Out*
  - while a queue is FIFO list, *First-In, First-Out*

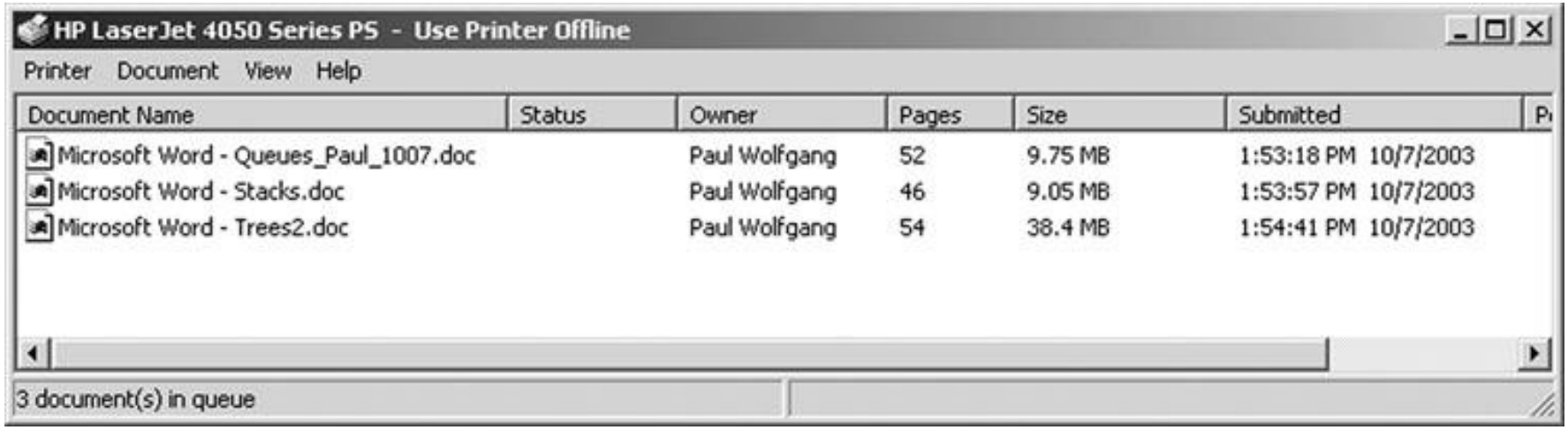
Queue Abstract Data Type

- A queue can be visualized as a line of customers waiting for service
- The next person to be served is the one who has waited the longest
- New elements are placed at the end of the line

Caryn J. Koffman

Print Queue

- Operating systems use queues to
  - keep track of tasks waiting for a scarce resource
  - ensure that the tasks are carried out in the order they were generated
- Print queue: printing is much slower than the process of selecting pages to print, so a queue is used to save files waiting to be printed.

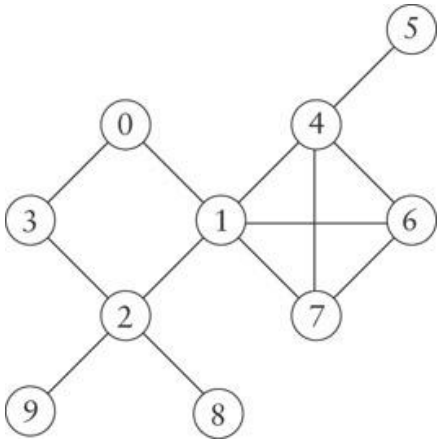


Unsuitability of a Print Stack

- Stacks are Last-In, First-Out (LIFO)
- The most recently selected document would be the next to print
- Unless the printer stack is empty, your print job may never be executed if others are issuing new print jobs

Using a Queue for Traversing a Multi-Branch Data Structure

- A graph models a network of nodes, with links connecting nodes to other nodes in the network
- A node in a graph may have several neighbors



- Programmers doing a *breadth-first traversal* often use a queue to ensure that nodes closer to the starting point are visited before nodes that are farther away
- You can learn more about graph traversal in Chapter 10

Specification for a Queue Interface

Method	Behavior
`boolean offer(E item)`	Inserts `item` at the rear of the queue. Returns **`true`** if successful; returns **`false`** if the item could not be inserted.
`E remove()`	Removes the entry at the front of the queue and returns it if the queue is not empty. If the queue is empty, throws a `NoSuchElementException`.
`E poll()`	Removes the entry at the front of the queue and returns it; returns **`null`** if the queue is empty.
`E peek()`	Returns the entry at the front of the queue without removing it; returns **`null`** if the queue is empty.
`E element()`	Returns the entry at the front of the queue without removing it. If the queue is empty, throws a `NoSuchElementException`.

- The `Queue` interface implements the `Collection` interface (and therefore the `Iterable` interface), so a full implementation of `Queue` must implement all required methods of `Collection` (and the `Iterable` interface)

Class `LinkedList` Implements the `Queue` Interface

- The `LinkedList` class provides methods for inserting and removing elements at either end of a double-linked list, which means all `Queue` methods can be implemented easily
- The `LinkedList` class implements the `Queue` interface

```
Queue<String> names = new LinkedList<>();
```

  - creates a new `Queue` reference, `names`, that stores references to `String` objects
  - The actual object referenced by `names` is of type `LinkedList<String>`, but because `names` is a type `Queue<String>` reference, you can apply only the `Queue` methods to it