IT 179
13
Queues

Reading Assignment

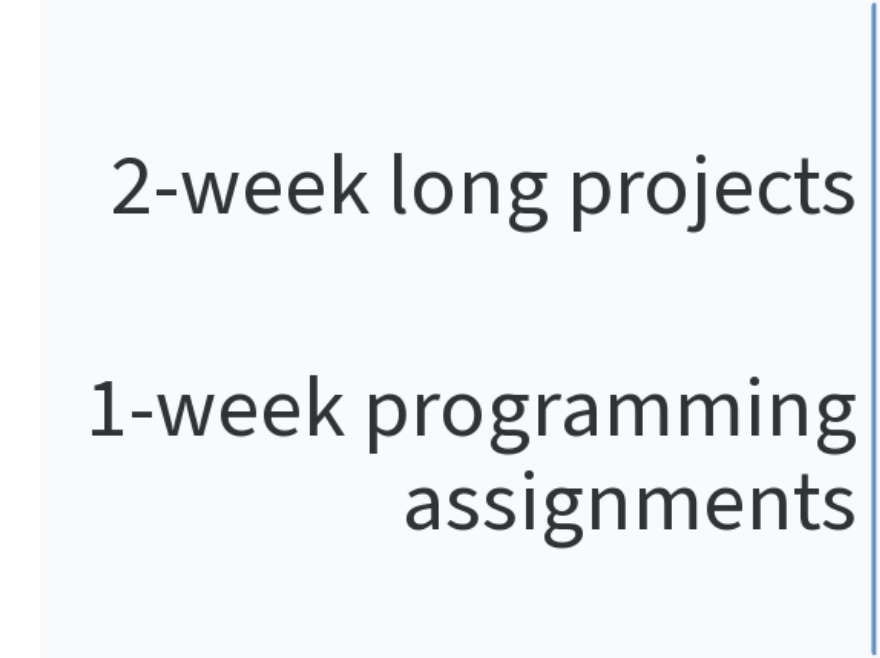
- Sections 4.5 - Queue Abstract Data Type
- Sections 4.6 - Queue Applications
- Sections 4.7 - Implementing the Queue Interface

Do you prefer live coding or explanation of prepared code?

Live Coding

Prepared Code

Do you prefer 2-week long projects or shorter 1-week programming assignments?



Abstract Data Types (ADT)

- An ADT is a type (or class) for objects whose **behavior** is defined by:
  - a set of value and
  - a set of operations.

Abstract Data Types (ADT)

- The definition of ADT
  - describes what operations can be performed on objects
  - does not describe how these operations will be implemented
  - does not specify how data will be organized in memory
  - does not specify what algorithms will be used for implementing the operations.

Abstract Data Types (ADT)

- Called “abstract” because it gives an implementation-independent view.

Name an ADT that we discussed before the Spring break.

Abstract Data Types (ADT) – Example 1

- List ADT
- Operations:
  - get() – Return an element from the list at any given position.
  - insert() – Insert an element at any position of the list.
  - remove() – Remove the first occurrence of any element from a non-empty list.
  - removeAt() – Remove the element at a specified location from a non-empty list.
  - replace() – Replace an element at any position by another element.
  - size() – Return the number of elements in the list.
  - isEmpty() – Return true if the list is empty, otherwise return false.
  - isFull() – Return true if the list is full, otherwise return false.

Abstract Data Types (ADT) – Example 2

- Stack ADT
- Operations
  - push() –
  - pop() –
  - peek() –
  - size() –
  - isEmpty() –
  - isFull() –

Queue ADT

- 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*

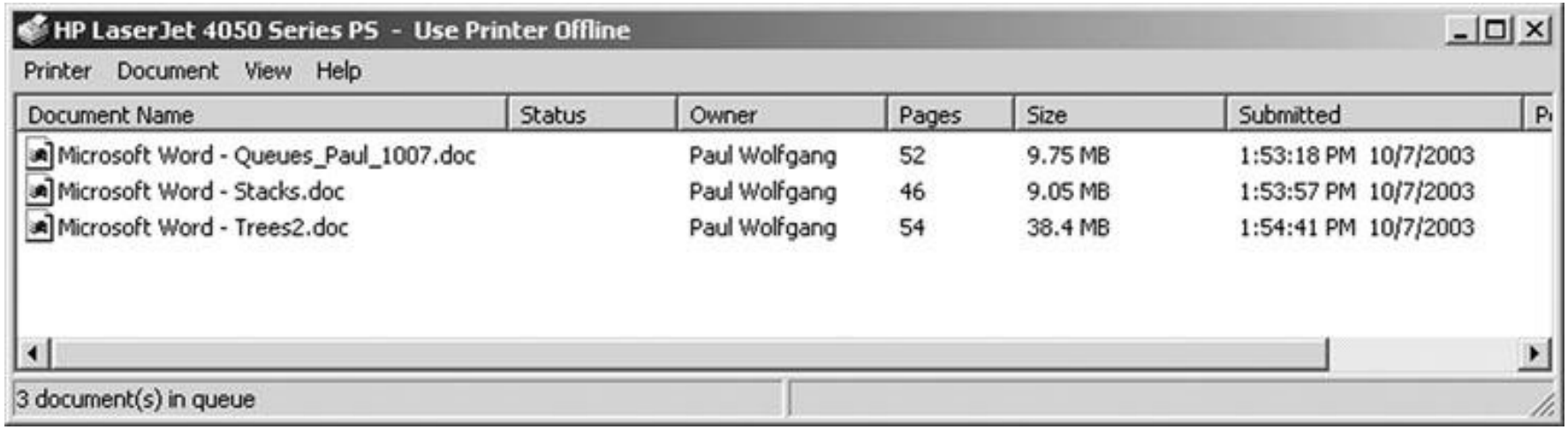
Queue Abstract Data Type

- 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

Queue

- public interface Queue<E> extends Collection<E>
- A collection used to hold multiple elements prior to processing.
- Besides basic Collection operations, a Queue provides these other operations:
  - insertion
  - extraction
  - inspection

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

Queue

- Demo
  - Reverse.java
    reverses the content of a stack

- Question
  - Can you reverse a stack without using queues?

Queue Application

Serving Two Queues of Customers

- Serve 2 customers who are **more than 50** years old
- Serve 1 customer who is **less than 50** years old
- Repeat