**QUEUES**

Queue interface has methods:

* for insertion
  1. offer
  2. add
* for removal
  1. remove
  2. poll
* for accessing the element at the front
  1. peek
  2. element

Queue is FIFO.

A picture containing text, silhouette

Description automatically generated

OS use queues to:

* keep track of tasks waiting for a limited 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

Graphical user interface, table

Description automatically generated

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 neighbours

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

Specification for a Queue Interface

Table

Description automatically generated with low confidence



Queue interface implements the Collection interface (therefore the Iterable interface)

So a full implementation of Queue must implement all required methods of Collection (and Iterable) interface

You can create a linked list and assign it to a variable type queue to use a queue without unnecessary methods (Check Collection framework).

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 Java 5.0 LinkedList class implements the Queue interface:  
 Queue<String> names = new LinkedList<String>();

* 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

Even if you define your queue like above, it is still iterable, it still has add and remove methods to add and remove an element anywhere in the queue. How can you make them inaccessible?

LinkedList is both List and Queue. For example add method has to be implemented in LinkedList. LinkedList may implement add method as an empty method.

How can you constraint on queue functionality if we can write our own class?

We can implement queue interface and keep LinkedList as a component. You can delegate queue’s method to delegate to LinkedList and handle them. You should also implement add and remove methods coming from Collection interface. Implement them and give an exception or sth like that.

Maintaining a Queue of Customers

Write a menu-driven program that maintains a list of customers

The user should be able to:

* insert a new customer in line (offer)
* display the customer who is next in line (peek)



* remove the customer who is next in line (poll)
* display the length of the line (size)
* determine how many people are ahead of a specified person (we can use iterator)



- Designing a Queue of Customers

Use JOptionPane.showOptionDialog() for the menu

Use a queue as the underlying data structure

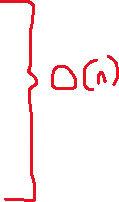
Write a MaintainQueue class which has a Queue<String> component customers

Graphical user interface, application

Description automatically generated

Text

Description automatically generated



If there is an infinite loop, then code is not an algorithm so processCustomers’ 3 line of code is not an algorithm bc that can be continue forever.

**Implementing the Queue Interface**

Using a Double-Linked List to Implement the Queue Interface

Insertion and removal from either end of a double-linked list is O(1) so either and can be the front (or rear) of the queue

Java designers decided to make the head of the linked list the front of the queue and the tail the rear of the queue.

Problem: If a LinkedList object is used as a queue, it will be possible to apply other LinkedList methods in addition to the ones required and permitted by the Queue interface

Solution: Create a new class with a LinkedList component and then code (by delegation to the LinkedList class) only the public methods required by the Queue interface

Using a Single-Linked List to Implement a Queue

Insertions are at the rear of a queue and removals are from the front

We need a reference to the last list node so that insertions can be performed at O(1)

The number of elements in the queue is changed by methods insert and remove

Diagram

Description automatically generated

DLL, SLL?

In terms of efficiency there is big difference, but you have to assign 2 references in DLL so SLL is a little better.

In terms of memory, SLL is better.

Using a Circular Array to Implement a Queue

The time efficiency of using a SLL or DLL to implement a queue is acceptable

However there are some space inefficiencies

Storage space is increased when using a linked list due to references stored in the nodes

Array implementation:

* Insertion at rear of array is constant time O(1)
* Removal from the front is linear time O(n)
* Removal from rear of array is constant time O(1)
* Insertion at the front is linear time O(n)

We now discuss how to avoid these inefficiencies in an array.

Diagram

Description automatically generatedDiagram

Description automatically generated

add

remove

Queue order now is like - - - > \* + / - A

If front is 1 more than the rear, either the queue is empty or queue is full.

Chart, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

offer is amortized constant running time.

Chart, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Box and whisker chart

Description automatically generated

Box and whisker chart

Description automatically generated

front and rear values are as in the empty queue but it is not bc size is 5.

Diagram, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Box and whisker chart

Description automatically generated

Diagram, box and whisker chart

Description automatically generated

Box and whisker chart

Description automatically generated

Chart

Description automatically generated with medium confidence



Table

Description automatically generated with medium confidence

Implementing Class ArrayQueue<E>.Iter

Just as for class ListQueue<E>, we must implement the missing Queue methods and an inner class Iter to fully implement Queue interface.

Text

Description automatically generatedText, letter

Description automatically generatedText

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

If you change public-private property, then you are overloading it.

Comparing the Three Implementations

Computation time

* All 3 implementations are comparable in terms of computation time
* All operations are O(1) regardless of implementation
* Although reallocating an array is O(n), it is amortized over n items, so the cost per item is O(1)

5 is constant, 1000 is constant as well. So one of them is better.

In SLL, we allocate memory for each item but in array we only do some assignments.

Storage

* Linked-list implementations require more storage due to the extra space required for the links
  + Each node for a SLL stores 2 references (one for the data, one for the link)
  + Each node for a DLL stores 3 references (one for the data, two for the links)
* A DLL requires 1.5 times the storage of a SLL
* A circular array that is filled to capacity requires half the storage of a SLL to store the same number of elements
  + but a recently reallocated circular array is half empty, and requires the same storage as a SLL

**The Deque Interface**

Short for double-ended queue

Allows insertions and removals from both ends

Java Collection Framework provides 2 implementations of the Deque interface

* ArrayDeque (Circular Array)
* LinkedList (DLL)

ArrayDeque uses a resizable circular array, but (unlike LinkedList) does not support indexed operations

ArrayDeque is the recommended implementation bc allocation in DLL takes time.

In time critical applications using ArrayDeque is not a good idea.

Table

Description automatically generated

All the operations are constant time.

Table

Description automatically generated

The Deque interface extends the Queue interface, so it can be used as a queue.

A deque can be used as a stack if elements are pushed and popped from the front of the deque

Using the Deque interface is preferable to using the legacy Stack class (based on Vector)

Table

Description automatically generated

**CASE STUDY**

Simulating Waiting Lines Using Queues

Simulation is used to study the performance of a physical system by using a physical, mathematical, or computer model of the system

Simulation allows designers of a new system to estimate the expected performance before building it

Simulation can lead to changes in the design that will improve the expected performance of the new system

Simulation is useful when the real system would be too expensive to build or too dangerous to experiment with after its construction

System designers often use computer models to simulate physical systems

* Example: an airline check-in counter

A branch of mathematics called queuing theory studies such problems

Diagram

Description automatically generated with medium confidenceBlue Skies Airlines (BSA) would like to have 2 waiting lines:

1. regular customers
2. frequent flyers

Assuming only one ticket agent, BSA would like to determine the average wait time for taking passengers from the waiting lines using various strategies:

* take turns serving passengers from both lines (one frequent flyer, one regular, one frequent flyer, etc.)
* serve the passenger waiting the longest
* serve any frequent flyers before serving regular passengers

Text

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, application

Description automatically generated

Diagram

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Text

Description automatically generated