# D:\C306 Data Structures and Algorithms\Problems\Draft\Problem 07 - Maple Story\Staff\queueimage.jpgQueue

Queue can be visualized as a queue in the lawn canteen for drinks.

A queue is a collections of items, all of the same type, where items are added to the tail of the queue and removed from the head of the queue.

Queue access stores data in a FIFO manner. FIFO – First in First out.

A queue is also an abstract data type.

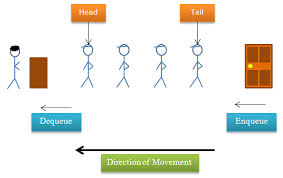
Two Functions used to manipulate the queue :

enqueue – Adding/Inserting an item to the queue to the **queue**.

dequeue – remove an item from the **queue**.

Elements are inserted at the rear/tail (enqueue) and removed from the back/head (dequeue)





<http://en.wikipedia.org/wiki/Queue_(data_structure)>

Read the above resource and answer the following questions:

1. What is a Queue?

|  |
| --- |
| A Queue is an example of a sequential collection |

1. How is a Queue different from Stack, ArrayList and Array?

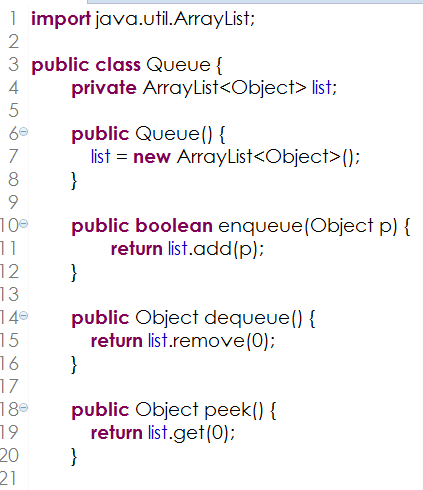
|  |
| --- |
| Queue = sequential access data structure  ArrayList, Array = random access data structure |

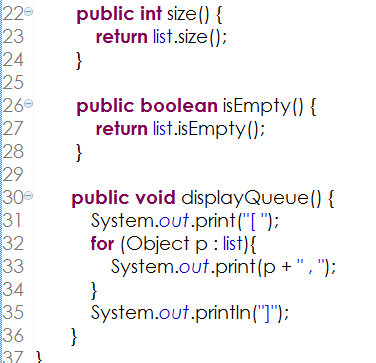
1. What are the operations that can be performed on a Queue?

|  |
| --- |
| Enqueuer  Front  Dequeuer  Is-full  Get-size |

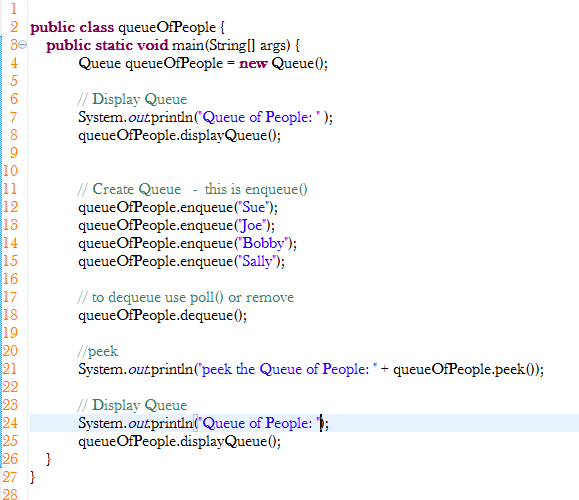
# Experimenting Queue class

Try the implementation of a queue and analyze the code.





A code below implements a queue of people waiting for their turn, using the queue class.



1. Write down the output of the above code segment.

|  |
| --- |
| Peek the queue of people: [“Sue”,”Joe”,”Bobby”,”Sally”] |

1. Write down the size of the queue at line number 16.

|  |
| --- |
| 4 |

1. Find out the difference between enqueue(item) and dequeue() method. Write down the return type for these methods.

|  |
| --- |
| Enqueue – to add an object to a queue  Dequeue – to remove an object to a queue |

1. Which method helps to remove the elements from the queue.

|  |
| --- |
| Dequeue |

1. Find out the difference between the remove element in a stack and a queue and their return data type

|  |
| --- |
| Stack – First in Last out  Queue – First in First out |

1. What is the element at the front and back of the queue after the code segment is executed.

|  |
| --- |
| Joe,Sally |

1. Add an another member (David) to the queue

|  |
| --- |
| queueOfPeople.enqueue(“David”) |

1. Add two lines of code at **line no 10**
   1. queueOfPeople.peek();
   2. queueOfPeople.dequeue();

Examine and write down the output of this code.

|  |
| --- |
|  |

1. Write a code to implement the queue using stack.

|  |
| --- |
| **import** java.util.Stack;  **public** **class** QueueStack {  **public** Stack<Object> data;  //constructor  **public** QueueStack(){  data = **new** Stack<Object>();  }  //enqueue  **public** **boolean** enqueue(Object value){  data.push(value);  **return** **true**;  }  //dequeue  **public** Object dequeue(){  Stack<Object>tempStack = **new** Stack<Object>();    **while** (!data.isEmpty()){  tempStack.push(data.pop()); // push the data that i pop from tempStack      }  //remove the front data in the queue  Object popedData = tempStack.pop();    **while** (!tempStack.isEmpty()){  data.push(tempStack.pop()); // push the data that i pop from tempStack  }  **return** popedData;  }  //size of the queue  **public** **int** size(){  **return** data.size();  }  //isEmpty  **public** **boolean** isEmpty(){  **return** data.isEmpty();  }  //peek  **public** Object front(){ // peek front of the queue  **return** data.get(0);  }  **public** **void** displayQueue(){  System.***out***.print("[ ");  **for** (Object value : data) //for value in object in data  System.***out***.print(value +", ");  System.***out***.print(" ]");  }  } |

1. Comparison of array, array, stack and queue

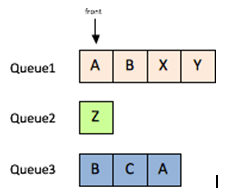
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Description** | **Array** | **ArrayLists** | **Stack** | **Queue** |
| Declare a variable that holds a list of numbers 1 to 5 | int[] list = new int[5]; | ArrayList<Interger> list = **new** ArrayList< Interger>(); | Stack<Integer> list = **new** Stack<Integer>(); | Queue list = new Queue(); |
| Add the numbers into the list | for (int i=0; i< 6; i++) {  list[i] = i+1;  } |  |  |  |
| Remove the element 4 from the list |  |  |  |  |
| Display the list |  |  |  |  |
| Get Size of the list |  |  |  |  |
| Get the Top / last item in the list |  |  |  |  |
| Get the First item in the list |  |  |  |  |

1. For the following scenario, fill in the appropriate queue’s operations.

|  |  |  |
| --- | --- | --- |
| **Num** | **Lawn Canteen Queue** | **Queue’s Operations** |
| 1 | Checking the number of people in the queue | Size |
| 2 | Person entering the queue | Enqueue |
| 3 | Person leaving the queue | Dequeue |
| 4 | Checking who is the 1st person in the queue | Peek |

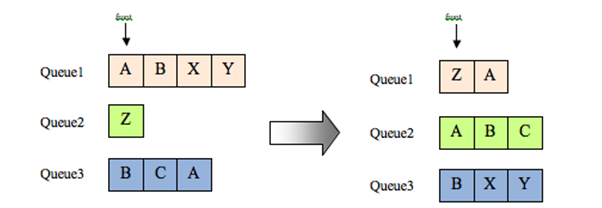
# Experimenting Queue class

The figure below shows three queues containing letters.



1. Write the code to create the above three queue using the Queue class

|  |
| --- |
| **public** **class** threeQueues {  **public** **static** **void** main(String[]args){  Queue queue1 = **new** Queue();  Queue queue2 = **new** Queue();  Queue queue3 = **new** Queue();  queue1.enqueue("A");  queue1.enqueue("B");  queue1.enqueue("X");  queue1.enqueue("Y");    queue2.enqueue("Z");    queue3.enqueue("B”);  queue3.enqueue("C”);  queue3.enqueue("A”);  }  } |
|  |



1. Using the appropriate queue operations, write code to move the letters among the three queues to produce the queues shown on the right in the figure above.

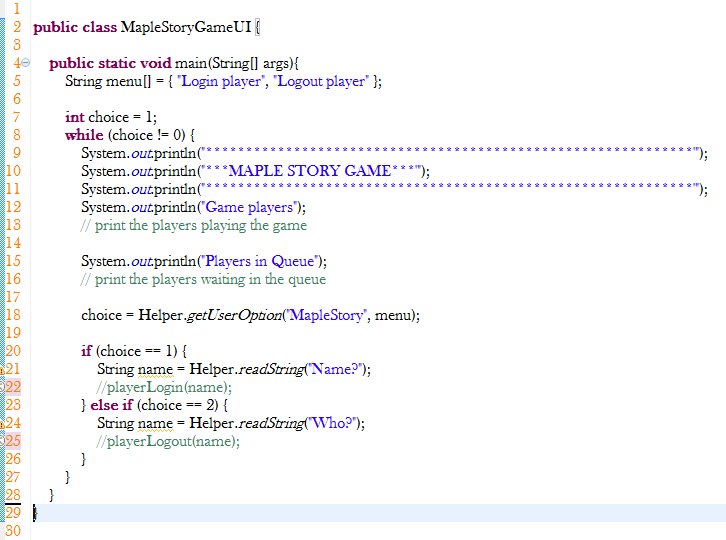
|  |  |  |
| --- | --- | --- |
| Sno | Transfer | Java Code |
| 1 | Move A from Queue1 to Queue2 | queue2.enqueue(queue1.dequeue()); |
| 2 | Move B from Queue3 to Queue2 | queue2.enqueue(queue3.dequeue()); |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 | Move Z from Queue2 to Queue1 | queue2.enqueue(queue2.dequeue()); |

1. Complete the codes to get the end result by using the above table to help you.

|  |
| --- |
|  |

# Modelling the Solution to the Problem

Create a project in Eclipse. Create a MapleStoryGameUI class, which displays the menu as shown below.



### Game Server Logging In

* The *Game Server* can be represented by an ArrayList of limited size.
* So when the 1st player logs in, the Player will enter the ArrayList
* This is repeated for the next 4 Player.
* Now when the next Player logs in, we check the *Game Server* is **FULL** and we print a message
* Let us implement the above.

### Game Server Logging out

* To logout a Player, we need to know the name of the Player who is logging out.
* Then we need to search through the *Game Server* for such a Player and remove him from it.
* Let us implement the above.

### Waiting Queue

* The Waiting Queue can be represented by a Queue
* Now when the next Player logs in and the *Game Server* is full, we place the Player in the *Waiting Queue*.
* When a Player logouts from the *Game Server*, then we should check the *Waiting Queue* to see if anyone is in it. If this is the case, then we should shift the first Player to the *Game Server*
* Let us implement the above.

## Enhancements

* What is the maximum length of the Queue used in the simulation?
* In real life, most people will not join a Queue when it is too long. Modify the codes to limit the queue size to a maximum of 10.
* Display a message when a player try to join the queue which is already at its maximum.

*End of Worksheet*