Implementing Stacks

**2**

|  |  |
| --- | --- |
| //this is the java.util implementation  **1**  public class ArrayListStack<E>  {  private ArrayList<E> a;  public ArrayListStack()   {    }  public E push(E obj) //O( )  {  }  public E pop() //O( )  {  }  public E peek() //O(  )  {     }  public boolean isEmpty() //O(  )  {  }  } | public class RawArrayStack  {  private Object[] a = new Object [10];  private int mySize = 0;  public RawArrayStack()   { }  public Object push(Object obj) //O( )  {    }  public Object pop() //O(  )  {  }  public Object peek() //O(  )  {     }  public boolean isEmpty() //O( )  {  }  }  **4** |
| public class LinkedListStack<E>   {  private LinkedList<E> ll;  public LinkedListStack<E>()   {     }  public E push(E obj) //O(  )  {  **3**  }  public E pop() //O(  )  {     }  public E peek () //O(  )  {     }  public boolean isEmpty() //O(  )  {  }  } | public class ListNodeStack<E>  {  private ListNode head;  public ListNodeStack<E>()   {  }  public E push(E obj) //O(  )  {  }  public E pop() //O(  )  {      }  public E peek() //O(  )  {  }  public boolean isEmpty() //O( )  {  }  } |

Implementing Queues

**2**

|  |  |
| --- | --- |
| public class ArrayListQueue<E>   {  private ArrayList<E> a;  public ArrayListQueue<E>()   {  **1**  }  public boolean add(E obj) //O(  )  {  }  public E remove() //O(  )  {  }  public E peek() //O(  )  {  }  public boolean isEmpty() //O(  )  {  }  } | public class RawArrayQueue<E>   {  private Object[] a = new Object[10];  private int mySize = 0; public RawArrayQueue()   { }  public boolean add(Object obj)//O( )  {  }  public Object remove() //O(  )  {  }  public Object peek() //O(  )  {     }  public boolean isEmpty() //O(  )  {  }  **4** |
| //this is the java.util implementation public class LinkedListQueue<E>   {  private LinkedList<E> ll;  public LinkedListQueue<E>()   {  **3**  }  public boolean add(E obj) //O(  )  {  }  public E remove() //O(  )  {  }  public E peek() //O(  )  {     }  public boolean isEmpty() //O( )  {  }  } | public class ListNodeQueue<E>  {  private ListNode head, tail;  public ListNodeQueue<E>()   {  }  public boolean add(E obj) //O()  {    **}**  public E remove() //O(  )  {  public E peek () //O(  )  {  }  public boolean isEmpty() //O(  )  {    } |

Implementing Priority Queues

as ArrayLists

|  |  |
| --- | --- |
| public class ArrayListPriorityQueue<E   extends Comparable<E>>  {  **1**  //elements in random order  private ArrayList<E> a; public ArrayListPriorityQueue<E>()   {     }  public boolean add(E obj) //O(  )  {  }  public E remove() //O(  )  {  }  public E peek() //O( )  {  }  public boolean isEmpty() //O( )  {  }  } | public class ArrayListPriorityQueue<E   extends Comparable<E>>  **2**  {  //elements sorted, Highest Priority at // the front private ArrayList<E> a; public ArrayListPriorityQueue<E>()   {     }  public boolean add (E obj) //O( )  {  }  public E remove() //O(  )  {     }  public E peek() //O(  )  {     }  public boolean isEmpty() //O( )  {  }  } |

Implementing Priority Queues

as LinkedLists

**4**

|  |  |
| --- | --- |
| **3**  public class LLPriorityQueue<E   extends Comparable<E>>  {  //elements in random order private LinkedList<E> ll;  public LLPriorityQueue<E>()   {   }  public boolean add(E obj) //O(  )  {  }  public E remove() //O(  )  {  }  public E peek() //O( )  {  }  public boolean isEmpty() //O(  )  {  }  }  **3** | public class LLPriorityQueue<E   extends Comparable<E>>  {  //elements sorted, Highest Priority at // the front  private LinkedList<E> ll; public LLPriorityQueue<E>()   {     }  public boolean add(E obj) //O( )  {  }  public E remove() //O(  )  {  }  public E peek() //O(  )  {     }  public boolean isEmpty() //O(  )  {  }  } |

N.B.: java.util.PriorityQueue is actually implemented by a *heap*. The heap's access time is O(log n).