-WeightIterator
-edges: Iterator<Edge>
-hasNext();
+boolean hasNext();
+Double next();
+void remove();
Iterator<Double>
#Edges
-vertex: VertexInterface<T>
-weight: double
#EdgeVertexInterface<T> endVertex, double edgeWeight;
#VertexInterface<T> getEndVertex();
-NeighborIterator
-edges: Iterator<Edge>
-NeighborIterator();
-boolean hasNext();
+VertexInterface<T> next();
+ void remove();
Iterator<VertexInterface<T>>
Vertex<T>
-label T
-edgeList: ListWithIteratorInterface<Edge>
-visited: boolean
-previousVertex: VertexInterface<T>
-cost: double
+Vertex(T vertexLabel);
+ T getLabel();
+boolean hasPredecessor();
+void setPredecessor(VertexInterface<T> predecessor);
+VertexInterface<T> getPredecessor();
+void visit();
+void unvisit();
+boolean isVisited();
+double getCost();
+void setCost(double newCost);
+String toString();
+Iterator<Double> getWeightIterator();
+boolean connect(VertexInterface<T> endVertex, double edgeWeight);
+boolean connect(VertexInterface<T> endVertex);
+Iterator<VertexInterface<T>> getNeighborIterator();
+boolean hasNeighbor();
+VertexInterface<T> getUnvisitedNeighbor();
+boolean equals(Object other);
-boolean disconnect(VertexInterface<T> endVertex);
+void display();
+LinkedListWithIterator<T>
-firstNode: Node
-numberOfEntries: int
+LinkedListWithIterator();
+void clear();
+void add(T newEntry);
+void add(int newPosition, T newEntry);
+T remove(int givenPosition, T newEntry);
+ T replace(int givenPosition, T newEntry);
+ T getEntry(int givenPosition);
+T[] toArray();
+boolean contains(T anEntry);
+int getLength();
+boolean isEmpty();
+Iterator<T> iterator();
+Iterator<T> getIterator();
-void initializeDataFields();
-Node getNodeAt(int givenPosition);
Iterable<T>
+ListWithIteratorInterface<T>
+Iterator<T> getIterator();
-Node
-data: T
-next: Node
-Node(T dataPortion);
-Node(T dataPortion, Node nextNode);
-T getData();
-void setData(T newData);
-Node getNextNode();
-void setNextNode(Node nextNode)
+ListInterface<T>
+void add(T newEntry);
+void add(int newPosition, T newEntry);
+T remove(int givenPosition);
+ void clear();
+T replace(int givenPosition, T newEntry);
+ T getEntry(int givenPosition);
+T[] toArray();
+boolean contains(T anEntry);
+int getLength();
+boolean isEmpty();
-IteratorForLinkedList
-nextNode: Node
-IteratorForLinkedList();
+ boolean hasNext();
+T next();
+void remove();
RuntimeException
EmptyQueueException
+ EmptyQueueException();
+ EmptyQueueException(String message)
+UndirectedGraph
-UndirectedGraph();
+boolean addEdge(T begin, T end, double edgeWeight);
+boolean addEdge(T begin, T end);
+int getNumberOfEdges();
Comparable<Entry PQ>
-EntryPQ
-vertex: VertexInterface<T>
-previousVertex: VertexInterface<T>
-cost: double
-EntryPQ(VertexInterface<T> vertex, double cost, VertexInterface<T> previousVertex);
+VertexInterface<T> getVertex();
+VertexInterface<T> getPredecessor();...
+ DirectedGraph<T>
-vertices: DictionaryInterface<T, VertexInterface<T>>
-edgeCount: int
+DirectedGraph();
+void removeVertex(T vertex);
+boolean removeEdge(T begin, T end);
#void reset(vertex);
+boolean hasVertex(T vertexLabel);
+QueueInterface<T> getBreadthFirstTraversal(T origin);
+QueueInterface<T> getDepthFirstTraversal(T origin);
+ StackInterface<T> getTopologicalOrder();
+int getShortestPath(T begin, T end, StackInterface<T> path);
+double getCheapestPath(T begin, T end, StackInterface<T> path);
#VertexInterface<T> findTerminal();
+void displayEdges();
+boolean hasVertex(T vertexLabel);
+boolean addEdge(T begin, T end, double edgeWeight);
+boolean addEdge(T begin, T end);
+boolean hasEdge(T begin, T end);
+ boolean isEmpty();
+int getNumberOfVertices();
+int getNumberOfEdges();
+void clear();
Driver
+static void main(String[] args);
+BasicGraphInterface<T>
+boolean addVertex(T vertexLabel);
+boolean addEdge(T begin, T end, double edgeWeight);
+boolean addEdge(T begin, T end);
+boolean hasEdge(T begin, T end);
+ boolean isEmpty();
+int getNumberOfVertices();
+int getNumberOfEdges();
+void clear();...
+GraphInterface<T>
+MaxHeap<T>
-heap: T[]
-lastIndex: int
-initialized= false: boolean
-DEFAULT_CAPACITY = 25: private static final int
MAX_CAPACITY = 10000: private static final int
+MaxHeap();
+MaxHeap(int initialCapacity);
+ MaxHeap(T[] entries);
+void add(T newEntry);
+T removeMax();
+T getMax();
+boolean isEmpty();
+int getSize();
+void clear();
-void reheapify(int rootIndex);
-void ensureCapacity();
-void checkInitialization();
-void checkCapacity(int capacity);
+MaxHeapInterface<T>
+void add(T newEntry);
+T removeMax();
+T getMax();
+boolean isEmpty();
+int getSize();
+void clear();
Comparable<? super T>>
+final class HeapPriorityQueue<T>
-pq: MaxHeapInterface<T>
+HeapPriorityQueue();
+void add(T newEntry);
+T remove();
+T peek();
+boolean isEmpty();
+int getSize();
+void clear();
+PriorityQueueInterface<T>
+void add(T newEntry);
+T remove();
+T peek();
+boolean isEmpty();
+int getSize();
+void clear();
Profile
-name: String
-image: String
-status: String
-friends: ArrayList<Profile>
+Profile(String name, String image, String status);
+String getName();
+String getImage();
+void setImage(String image);
+String getStatus();
+void setStatus(String status);
+ArrayList<Profile> getFriends();
+void addFriend(Profile friend);
+void removeFriend(Profile friend);
+SocialNetwork
-profiles: ArrayList<Profile>
-profileDictionary: LinkedDictionary<String, Profile>
+SocialNetwork();
+void joinNetwork(String name, String image, String status);
+void leaveNetwork(String name);
+void updateProfile(String name, String newImage, String newStatus);
+void addFriend(String name, String friendName);
+void removeFriend(String name, String friendName);
+void searchProfile(String name);
+void displayNetwork();
-Node
-key: K
-value: V
-next: Node
-Node(K searchKey, V dataValue);
-Node(K searchKey, V dataValue, Node nextNode);
-K getKey();
-V getValue();
-void setValue(V newValue);
-Node getNextNode();
-void setNextNode(Node nextNode);
+LinkedDictionary<K, V>
-firstNode: Node
-numberOfEntries: int
+ LinkedDictionary();
+V add(K key, V value);
+V remove(K key);
+V getValue(K key);
+ boolean contains(K key);
+Iterator<K> getKeyIterator();
+Iterator<V> getValueIterator();
+boolean isEmpty();
+ int getSize();
+void clear();
+DictionaryInterface<K, V>
+V add(K key, V value);
+V remove(K key);
+V getValue(K key);
+ boolean contains(K key);
+Iterator<K> getKeyIterator();
+Iterator<V> getValueIterator();
+boolean isEmpty();
+ int getSize();
+ void clear();...
KeyIterator
-nextNode: Node
+KeyIterator();
+boolean hasNext();
+K next();
+void remove();
Iterator<K>
ValueIterator
-nextNode: Node
-ValueIterator();
+boolean hasNext();
+V next();
+void remove();
Iterator<V>
+GraphAlgorithmsInterface<T>
+QueueInterface<T> getBreadthFirstTraversal(T origin);
+QueueInterface<T> getDepthFirstTraversal(T origin);
+StackInterface<T> getTopologicalOrder();
-int getShortestPath(T begin, T end, StackInterface<T> path);
+double getCheapestPath(T begin, T end, StackInterface<T> path);
+StackInterface<T>
+void push(T newEntry);
+T pop();
+T peek();
+boolean isEmpty();
+void clear();
+final class LinkedStack<T>
-topNode: Node
+LinkedStack();
+void push(T newEntry);
+T pop();
-T peek();
+boolean isEmpty();
+void clear();
-Node
-data: T
-next: Node
-Node(T data, Node next);
+QueueInterface<T>
+void enqueue(T newEntry);
+T dequeue();
+T getFront();
+boolean isEmpty();
+void clear();
+final class LinkedQueue<T>
-firstNode: Node
-lastNode: Node
+LinkedQueue();
+void enqueue(T newEntry);
+T dequeue();
-void checkInitialization();
+T getFront();
+boolean isEmpty();
+void clear();
Node
-data: T
-next: Node
-Node(T dataPortion);
-Node(T dataPortion, Node nextNode);
-T getData();
-void setData(T newData);
-Node getNextNode();
-void setNextNode(Node nextNode);