An AssociationMap is an abstract data type that maintains a collection of Association objects. An Association is an object that has two variables: key and data. It records the fact that key is associated with data. For example, key might be a person’s name and data might be the person’s phone number. A telephone book could be implemented using an AssociationMap that maintains a collection of Associations between names and phone numbers. In this laboratory exercise, you will implement the Association class; the AssociationMap interface, and a class AssociationMapBST, which uses a binary search tree to implement the AssociationMap interface. When you are done, you will be able to use the class AssociationMapDemo to test your implementation by performing operations on an AssociationMap of names and numbers.

1. Implement the generic Association<K extends Comparable<K>, D> class which implements the Comparable<Association<K,D>> interface. The class has two public variables: key (of type K) and data (of type D). It also has the following three public methods:
   - Association(K key) Initializes this.key to key and this.data to null.
   - Association(K key, D data) Initializes this.key to key and this.data to data.
   - int compareTo(Association<K,D> other) Uses the compareTo method of the K data type to compare the key of this association to the key of other and returns the result.
   - String toString() Generates the string “<key,data>”, were key and data are the actual values of this.key and this.data.

2. Implement the generic AssociationMap<K extends Comparable<K>, D> interface, which has the following public methods:
   - insert takes key of type K and data of type D and returns nothing.
   - delete takes key of type K and returns an object of type D.
   - lookUp takes key of type K and returns an object of type D.
   - elements takes no arguments and returns a Queue<Association<K,D>> object.

3. Implement the generic AssociationMapBST<K extends Comparable<K>, D> class, which implements the AssociationMap<K,D>> interface. This class has a private variable bsTree of type BSTree<Association<K,D>>, and the following public methods:
   - AssociationMapBST() Initializes bsTree to be an empty binary search tree.
   - void insert(K key, D data) Constructs an Association between key and data and then inserts the association into bsTree.
   - D delete(K key) Constructs an Association between key and nothing. It then invokes the delete method of bsTree, passing the Association as the parameter. The compareTo method of Association has been defined so that bsTree’s delete method will delete any Association with this key. Furthermore, bsTree’s delete method will return the association that was deleted, or will return null if key does not appear in bsTree. The delete method of AssociationMapBST should return the data in the Association that was deleted, or else return null if nothing was deleted.
   - D lookUp(K key) Constructs an Association between key and nothing. It then invokes the member method of bsTree, passing this Association as the parameter. The compareTo method of Association has been defined so that bsTree’s member method will return any Association with this key, or will return null if key does not appear in bsTree. The lookUp method of AssociationMapBST should return the data in the Association that was found, or else return null if nothing was found.
   - Queue<Association<K,D>> elements() Invokes the elements method of bsTree to return a queue that provides sequential access to the data in the binary search tree.

4. Run the AssociativeMapDemo main class to test your implementation.