CS2400 - Data Structures and Advanced Programming
Module 13: Dictionaries

Hao Ji

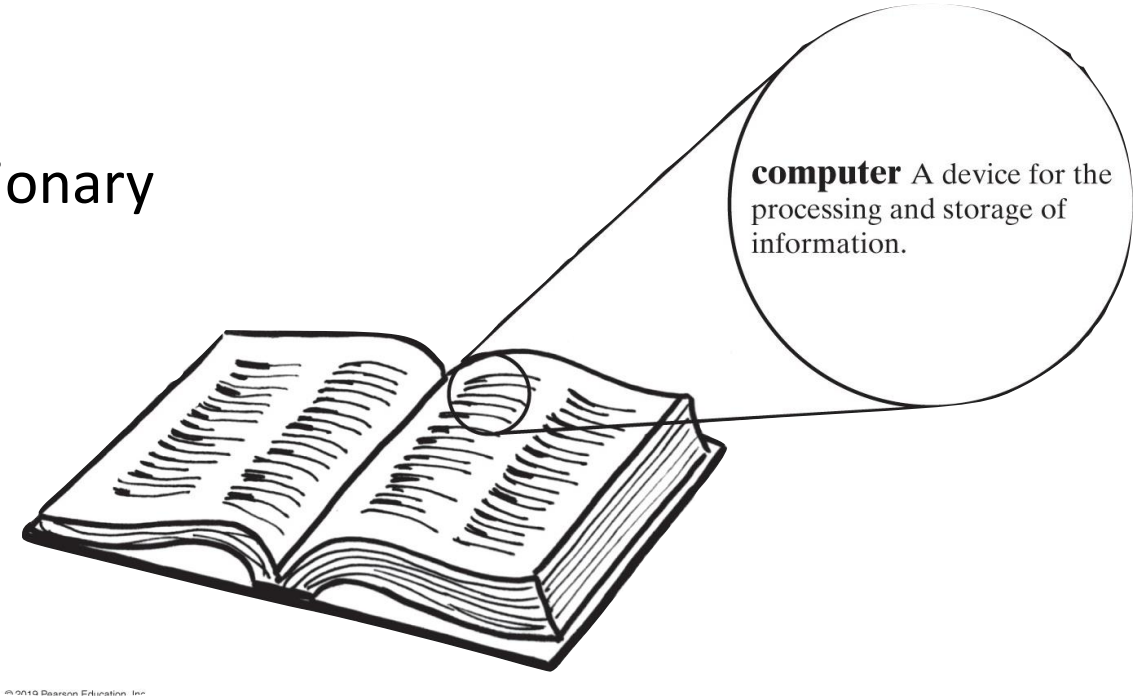
Computer Science Department

Cal Poly Pomona

Dictionaries

- When you want to look up …
  - The meaning of a word
  - An address
  - A phone number

- These can be implemented in an ADT Dictionary

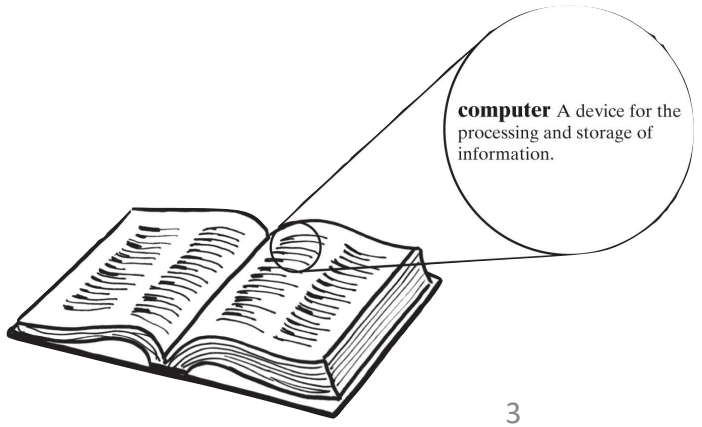


Dictionaries

- The ADT **dictionary** is also called a **map**, **table**, or **associative array**.

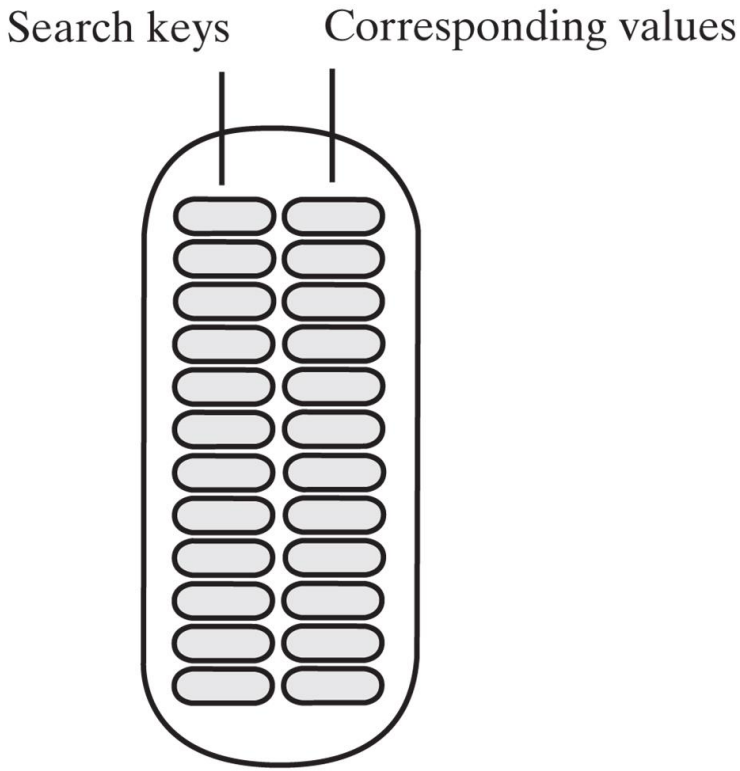
- The dictionary contains entries that each have two parts
  - Keyword, search key
  - Value

- The search key enables you to locate the desired entry



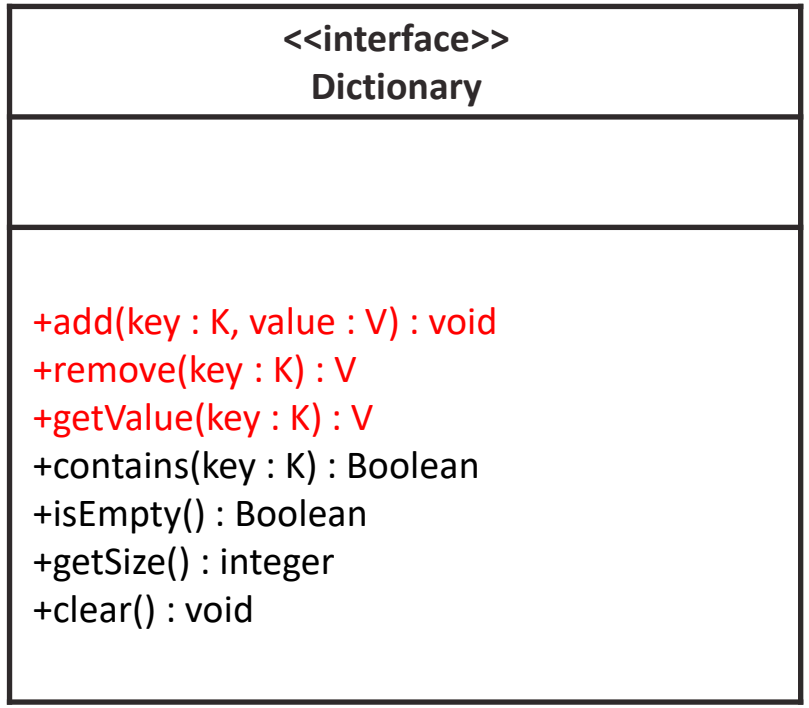
Dictionaries

- Dictionary Data
  - Collection of pairs `(k, v)` of objects `k` and `v`,
    - *k* is the search key
    - *v* is the corresponding value
  - *Number of pairs* in the collection



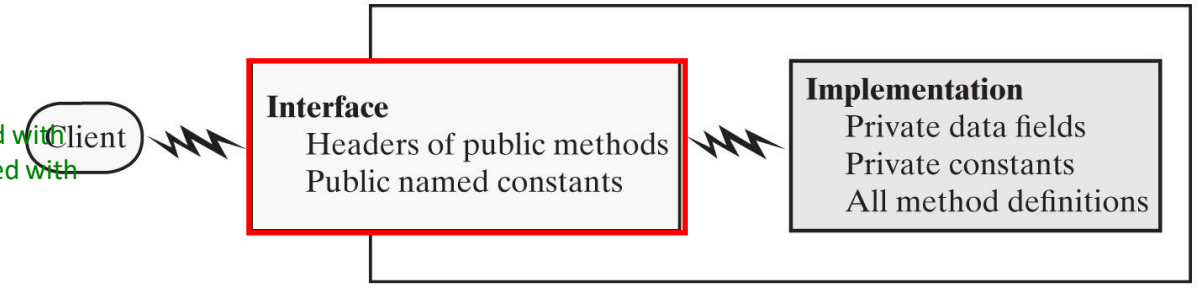
A dictionary object

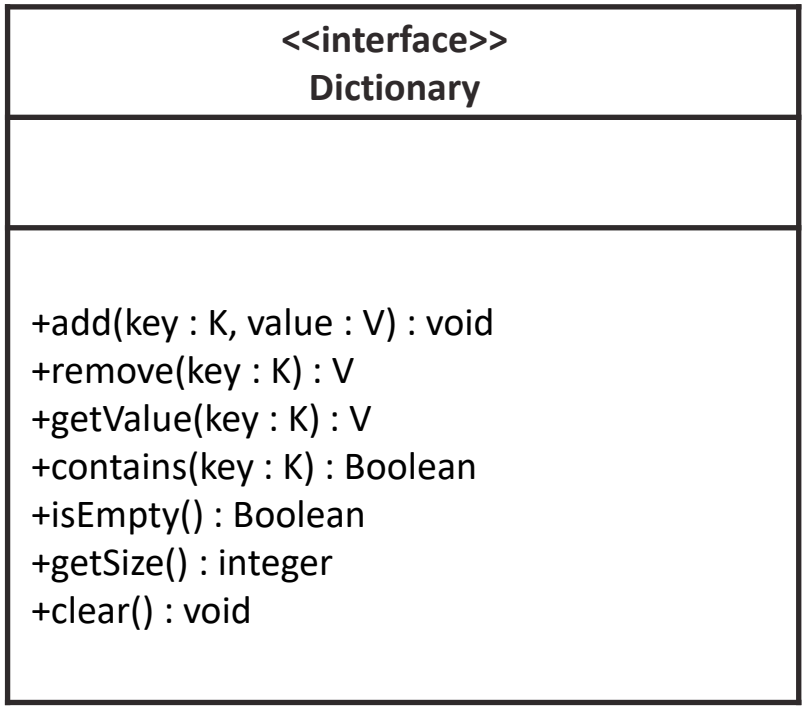
Using UML Notation to Specify a Class



// Adds the pair (key, value) to the dictionary
// Removes from the dictionary the entry that corresponds to a given search key
// Retrieves from the dictionary the value that corresponds to a given search key
// Sees whether any entry in the dictionary has a given search key.
// Sees whether the dictionary is empty
// Gets the size of the dictionary
// Removes all entries from the dictionary

```
/** An interface for a dictionary with distinct search keys.  Search keys and associated values are not null. */
public interface DictionaryInterface<K, V>
{
  /** Adds a new entry to this dictionary. If the given search key already exists in the dictionary, replaces the
corresponding value. @param key    An object search key of the new entry. @param value  An object associated with
the search key. @return  Either null if the new entry was added to the dictionary or the value that was associated with
key if that value was replaced. */
  public V add(K key, V value);

  /** Removes a specific entry from this dictionary. @param key  An object search key of the entry to be removed.
@return  Either the value that was associated with the search key or null if no such object exists. */
  public V remove(K key);

  /** Retrieves from this dictionary the value associated with a given search key. @param key  An object search key of
the entry to be retrieved.  @return  Either the value that is associated with the search key or null if no such object
exists. */
  public V getValue(K key);

 /** Sees whether a specific entry is in this dictionary. @param key  An object search key of the desired entry.
@return  True if key is associated with an entry in the dictionary. */
  public boolean contains(K key);

/** Sees whether this dictionary is empty.  @return  True if the dictionary is empty. */
  public boolean isEmpty();

  /** Gets the size of this dictionary. @return  The number of entries (key-value pairs) currently in the dictionary. */
  public int getSize();

  /** Removes all entries from this dictionary. */
  public void clear();
} // end DictionaryInterface
```





Implementations of a Dictionary

- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary
- Linked Implementations
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary

Implementations of a Dictionary

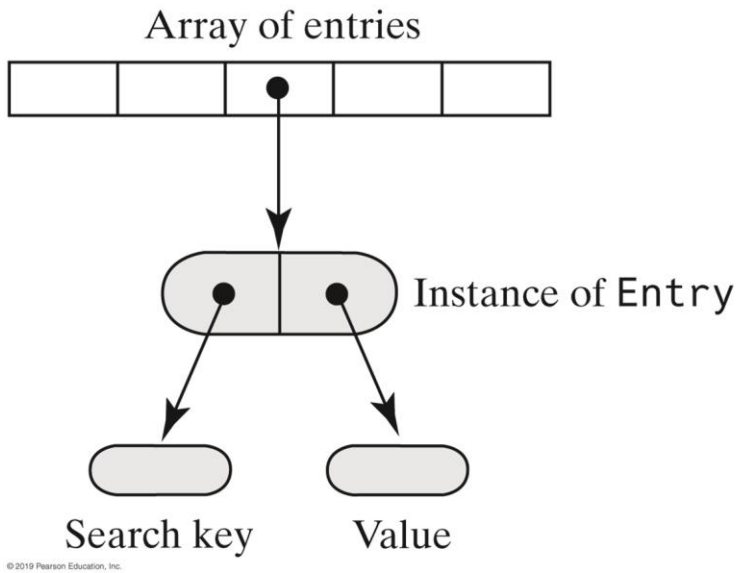
- **Array-Based Implementations**
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary

- Linked Implementations
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary

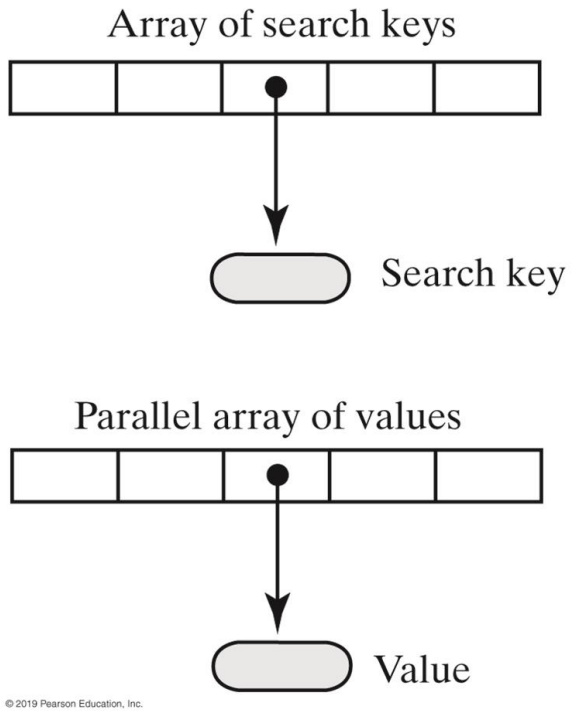
Implementations of a Dictionary

- Two possible ways to use arrays to represent the entries in a dictionary

(a) An array of objects that encapsulate each search key and corresponding value



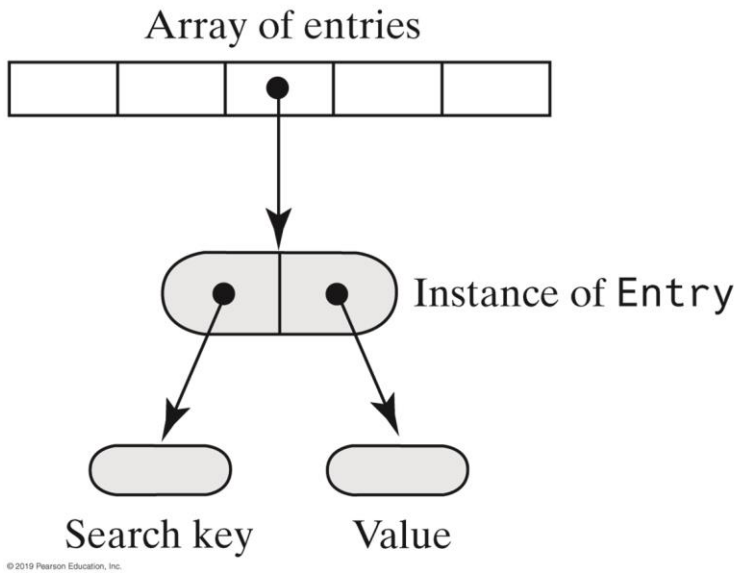
(b) Two arrays in parallel, one of search keys and one of values



Implementations of a Dictionary

- Two possible ways to use arrays to represent the entries in a dictionary

(a) An array of objects that encapsulate each search key and corresponding value



(b) Two arrays in parallel, one of search keys and one of values

Implementations of a Dictionary

- **Array-Based Implementations**
  - **An Unsorted Array-Based Dictionary**
  - A Sorted Array-Based Dictionary
- Linked Implementations
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary
- Vector-Based Implementations

AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Implementations of a Dictionary

```
/** A class that implements the ADT dictionary by using a resizable array.
  The dictionary is unsorted and has distinct search keys.
  Search keys and associated values are not null.  */
public class ArrayDictionary<K, V> implements DictionaryInterface<K, V>
{
        private Entry<K, V>[] dictionary; // Array of unsorted entries
        private int numberOfEntries;
        private boolean integrityOK = false;
        private final static int DEFAULT_CAPACITY = 25;
        private static final int MAX_CAPACITY = 10000;

        public ArrayDictionary() {
                this(DEFAULT_CAPACITY);
        } // end default constructor

        public ArrayDictionary(int initialCapacity)
        {
                checkCapacity(initialCapacity);

                // The cast is safe because the new array contains null entries
                @SuppressWarnings("unchecked")
                Entry<K, V>[] tempDictionary = (Entry<K, V>[])new Entry[initialCapacity];
                dictionary = tempDictionary;
                numberOfEntries = 0;
                integrityOK = true;
        } // end constructor

/* < Implementations of methods in DictionaryInterface. >  . . . */
```

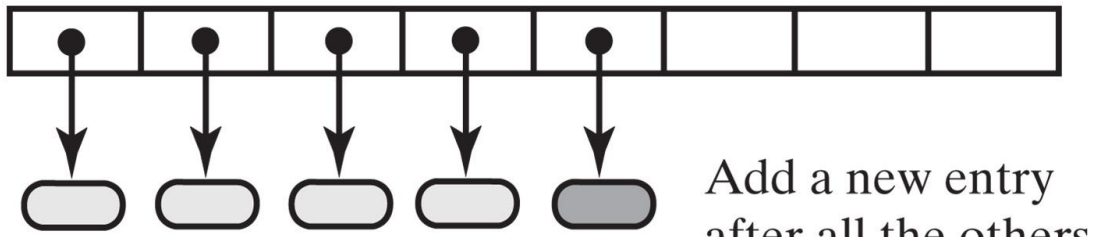
```
        private class Entry<K, V>
        {
                private K key;
                private V value;

                private Entry(K searchKey, V dataValue)
                {
                    key = searchKey;
                    value = dataValue;
                } // end constructor

                private K getKey()
                {
                        return key;
                } // end getKey

                private V getValue()
                {
                        return value;
                } // end getValue

                private void setValue(V dataValue)
                {
                        value = dataValue;
                } // end setValue
        } // end Entry
} // end ArrayDictionary
```

AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Unsorted Array-Based Implementations

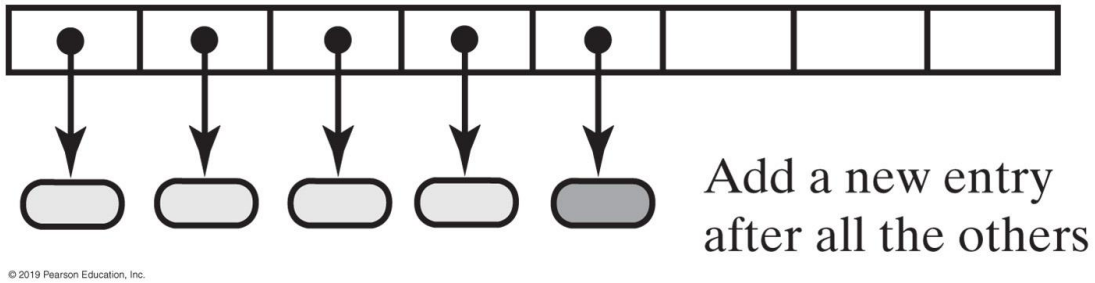
- **Array-Based Implementations**
  - **An Unsorted Array-Based Dictionary**

- Adding an entry
  - **Case 1:** If key doesn't exist in the dictionary, adds a new key-value entry to the dictionary (after the last one).
  - **Case 2:** If key already exists in the dictionary, returns the corresponding value and replaces it with value. (Note that key and value are not null)



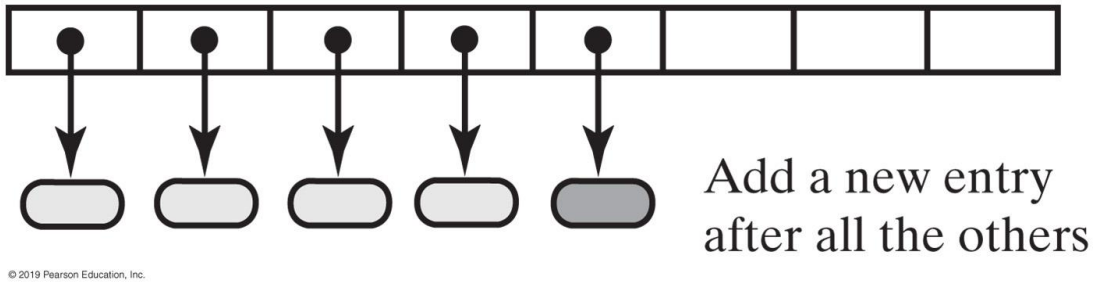
AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

```
public V add(K key, V value)
{
  checkIntegrity();
  if ((key == null) || (value == null))
    throw new IllegalArgumentException("Cannot add null to this dictionary.");
  else
  {
    V result = null;
    int keyIndex = locateIndex(key);
    if (keyIndex < numberOfEntries)
    {
      // Key found, return and replace entry's value
      result = dictionary[keyIndex].getValue(); // Get old value
      dictionary[keyIndex].setValue(value);              // Replace value
    }
    else // Key not found; add new entry to dictionary
    {
      // Add at end of array
      dictionary[numberOfEntries] = new Entry<>(key, value);
      numberOfEntries++;
      ensureCapacity(); // Ensure enough room for next add
    } // end if
    return result;
  } // end if
} // end add
```



AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

```java
public V add(K key, V value)
{
  checkIntegrity();
  if ((key == null) || (value == null))
    throw new IllegalArgumentException("Cannot add null to this dictionary.");
  else
  {
    V result = null;
    int keyIndex = locateIndex(key);
    if (keyIndex < numberOfEntries)
    {
      // Key found, return and replace entry's value
      result = dictionary[keyIndex].getValue(); // Get old value
      dictionary[keyIndex].setValue(value);              // Replace value
    }
    else // Key not found; add new entry to dictionary
    {
      // Add at end of array
      dictionary[numberOfEntries] = new Entry<>(key, value);
      numberOfEntries++;
      ensureCapacity(); // Ensure enough room for next add
    } // end if
    return result;
  } // end if
} // end add
```



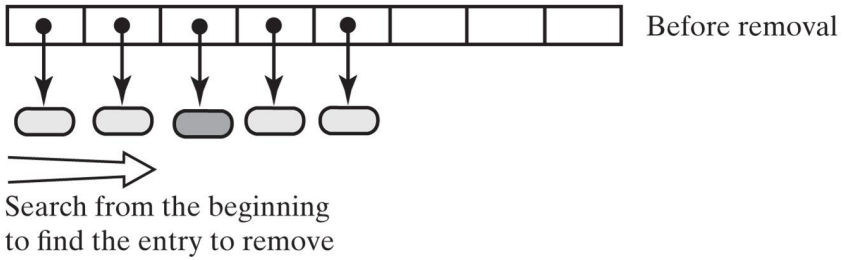
AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

To search an unsorted array

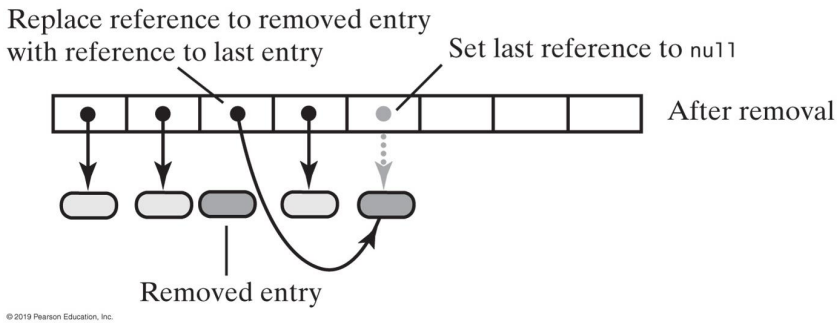
```java
private int locateIndex(K key)
{
      // Sequential search
      int index = 0;
      while ( (index < numberOfEntries) && !key.equals(dictionary[index].getKey()) )
            index++;
      return index;
}
```

Unsorted Array-Based Implementations

- Removing an entry from an unsorted array-based dictionary
  - We first locate the entry, and



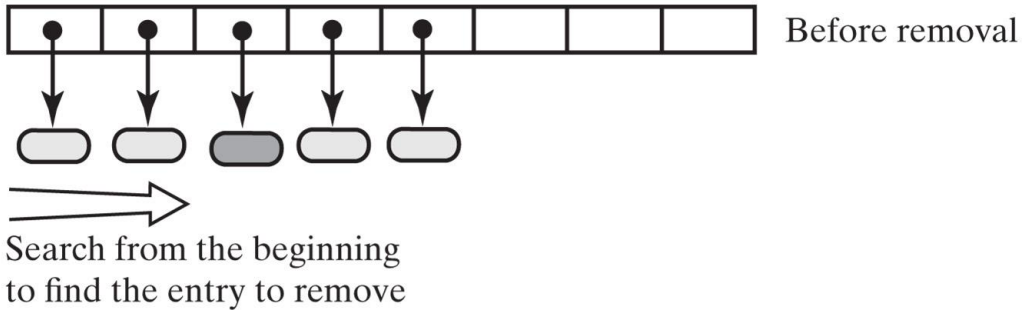
  - then replace it with **the last entry** in the dictionary

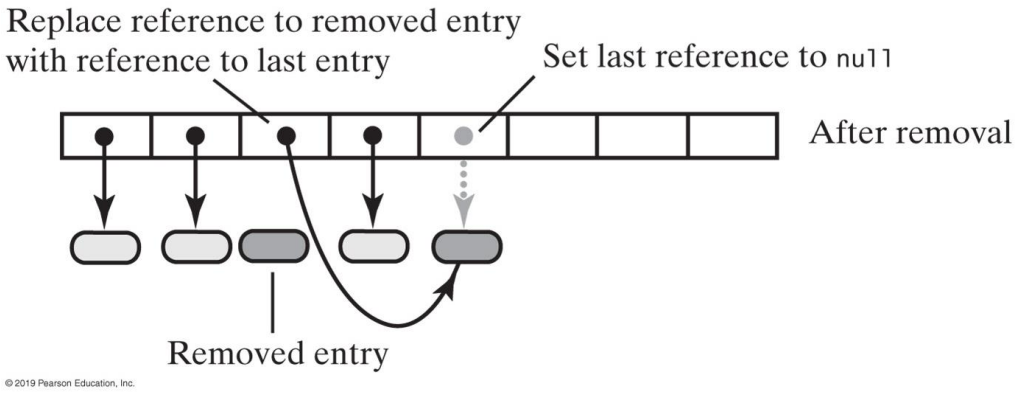


AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void **+remove(key : K) : V** +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

In-Class Exercises

- Write the method *public V remove(K key)*

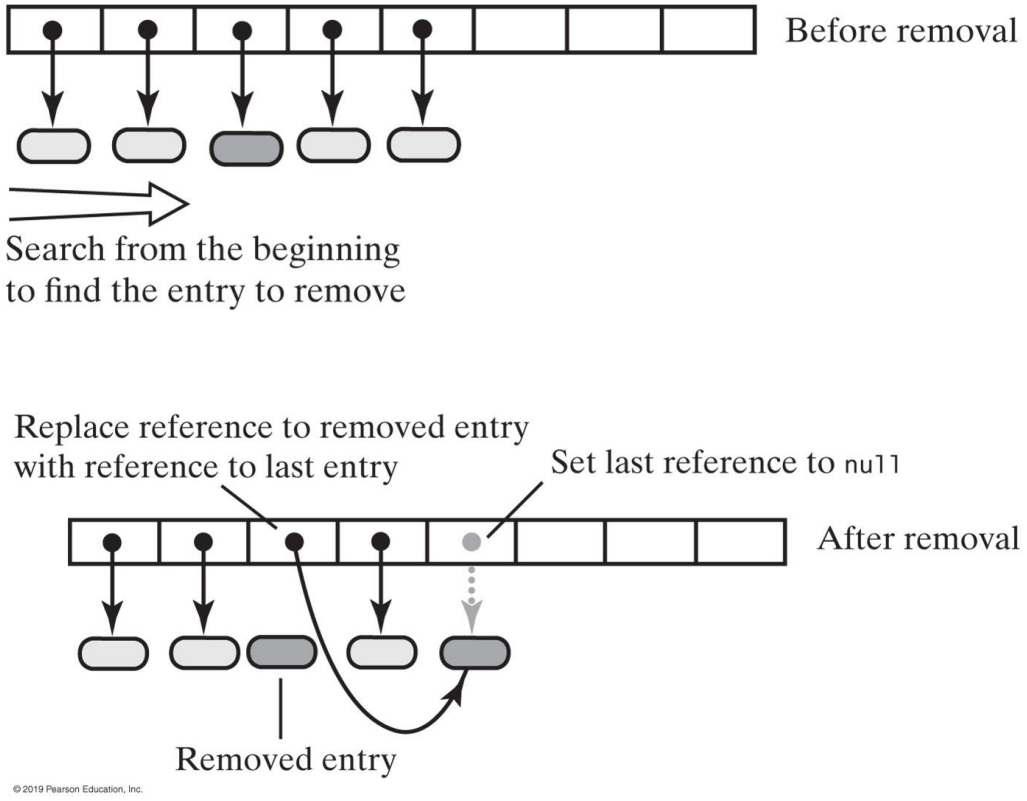




In-Class Exercises

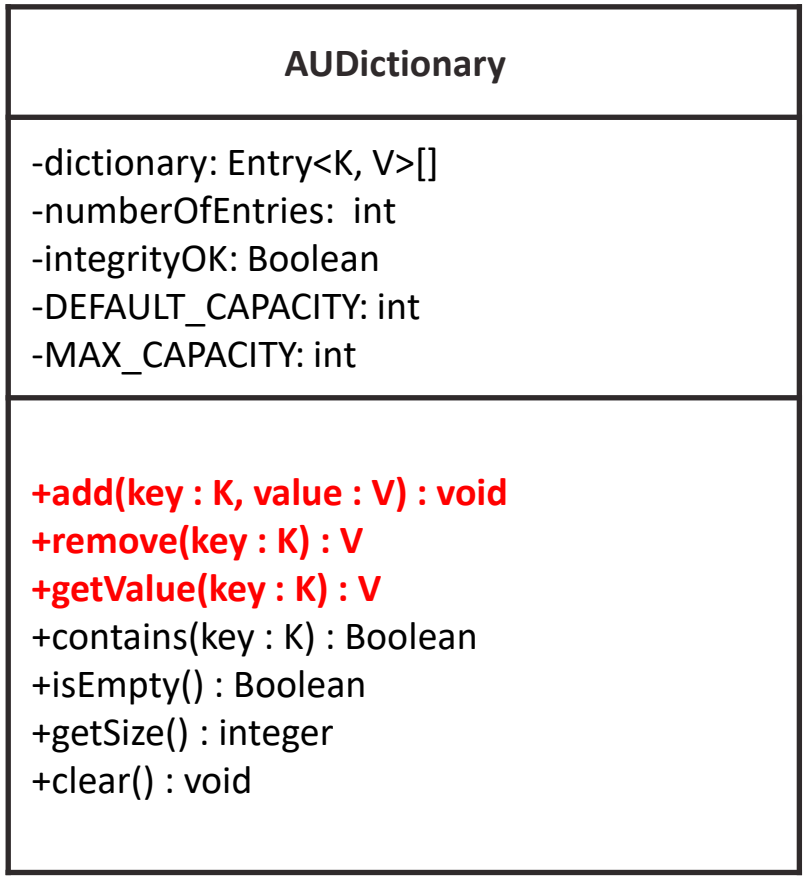
- Write the method *public V remove(K key)*

```
public V remove(K key)
{
   checkIntegrity();
   V result = null;
   int keyIndex = locateIndex(key);

   if (keyIndex < numberOfEntries)
   {
        // Key found; remove entry and return its value
        result = dictionary[keyIndex].getValue();
        // Replace removed entry with last entry
        dictionary[keyIndex] = dictionary[numberOfEntries - 1];
        dictionary[numberOfEntries - 1] = null;
        numberOfEntries--;
   } // end if
   // Else result is null

   return result;
} // end remove
```



Unsorted Array-Based Implementations

- What is the **Big Oh** of each dictionary method in the worst case?



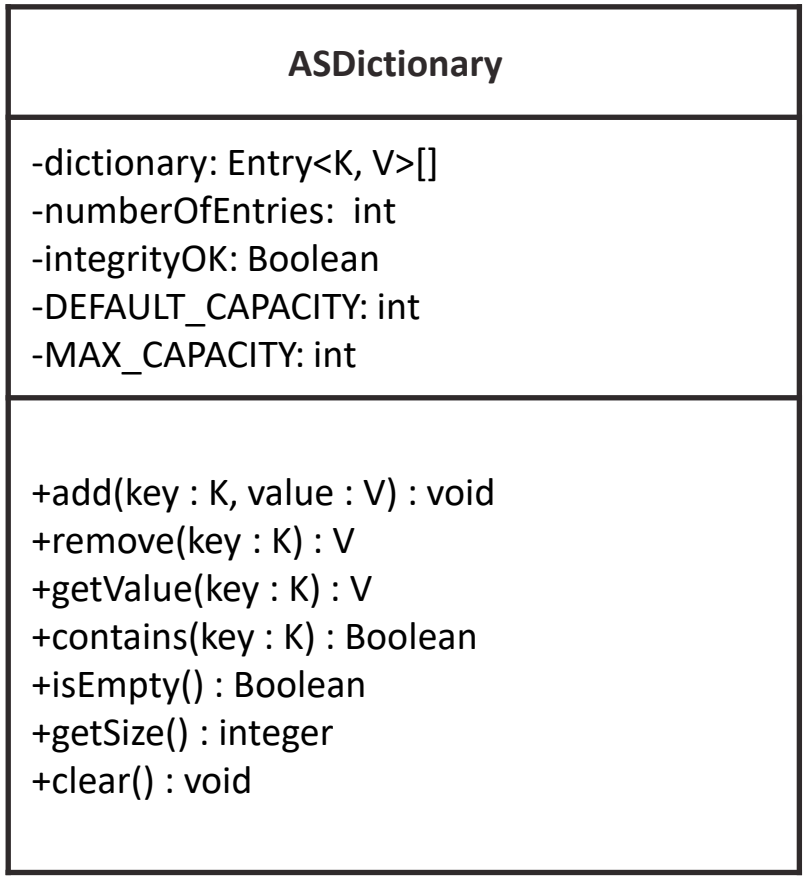
Unsorted Array-Based Implementations

- What is the **Big Oh** of each dictionary method in the worst case?
  - Addition: $O(n)$
  - Removal: $O(n)$
  - Retrieval: $O(n)$

AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void **+remove(key : K) : V** **+getValue(key : K) : V** +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Implementations of a Dictionary

- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - **A Sorted Array-Based Dictionary**
- Linked Implementations
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary



Implementations of a Dictionary

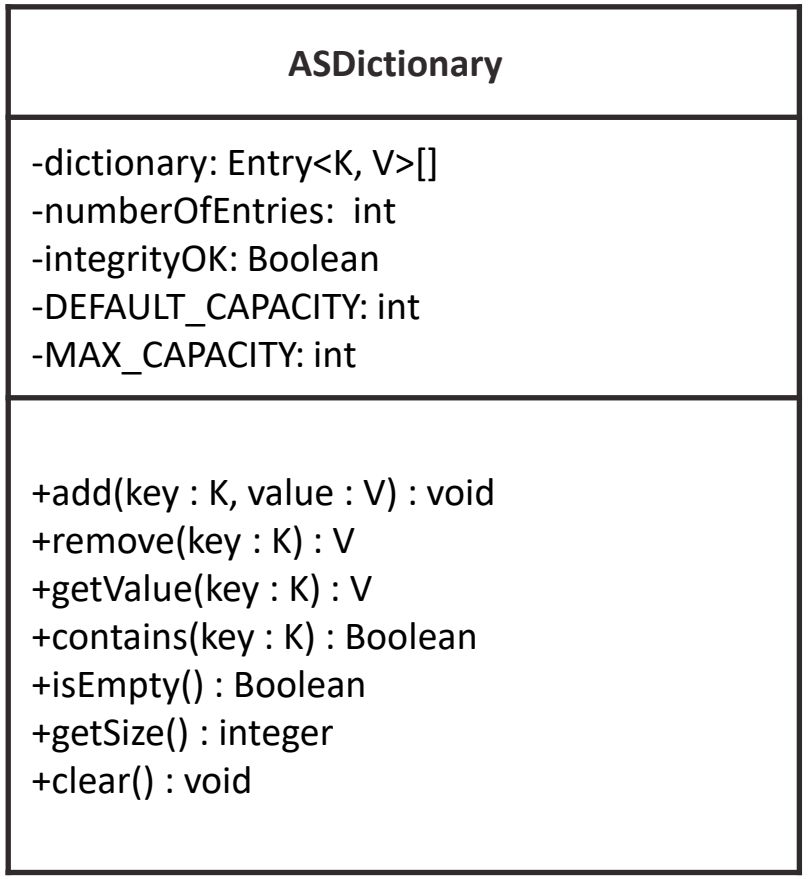
```
/** A class that implements the ADT dictionary by using a resizable array.
   The dictionary is sorted and has distinct search keys. Search keys and
   associated values are not null. */
public class SortedArrayDictionary<K extends Comparable<? super K>, V>
         implements DictionaryInterface<K, V>
{
  // Array of entries sorted by search key
        private Entry<K, V>[] dictionary;
        private int numberOfEntries;
        private boolean integrityOK = false;
        private final static int DEFAULT_CAPACITY = 25;
        private static final int MAX_CAPACITY = 10000;

/* < Constructors analogous to those in Listing 21-1.
    . . .
    Implementations of methods in DictionaryInterface.
     . . .
    The private class Entry, as shown in Listing 21-1.
    . . .
 */
} // end SortedArrayDictionary
```

ASDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Implementations

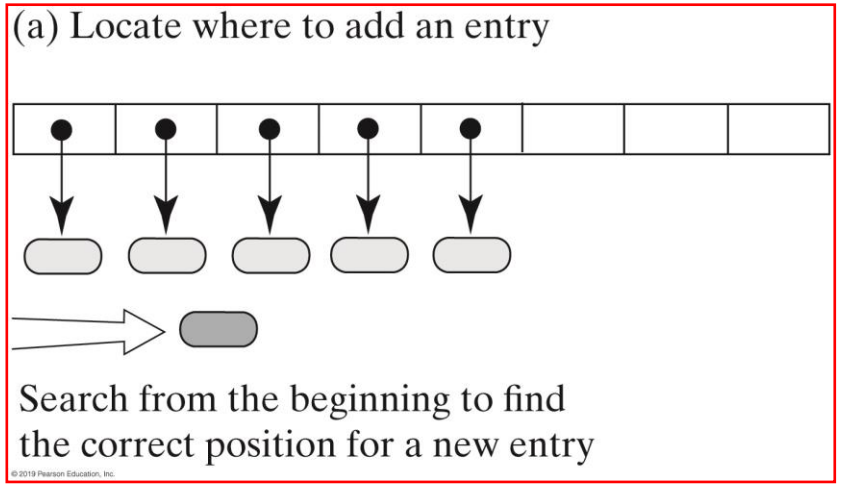
The notation **K extends Comparable<? super K>**, defines the generic type K. It allows us to compare objects of type K with either objects of type K or objects of any superclass of K

```
/** A class that implements the ADT dictionary by using a resizable array.
   The dictionary is sorted and has distinct search keys. Search keys and
   associated values are not null. */
public class SortedArrayDictionary<K extends Comparable<? super K>, V>
         implements DictionaryInterface<K, V>
{
  // Array of entries sorted by search key
        private Entry<K, V>[] dictionary;
        private int numberOfEntries;
        private boolean integrityOK = false;
        private final static int DEFAULT_CAPACITY = 25;
        private static final int MAX_CAPACITY = 10000;

/* < Constructors analogous to those in Listing 21-1.
    . . .
    Implementations of methods in DictionaryInterface.
     . . .
    The private class Entry, as shown in Listing 21-1.
    . . .
 */
} // end SortedArrayDictionary
```



Sorted Array-Based Implementations

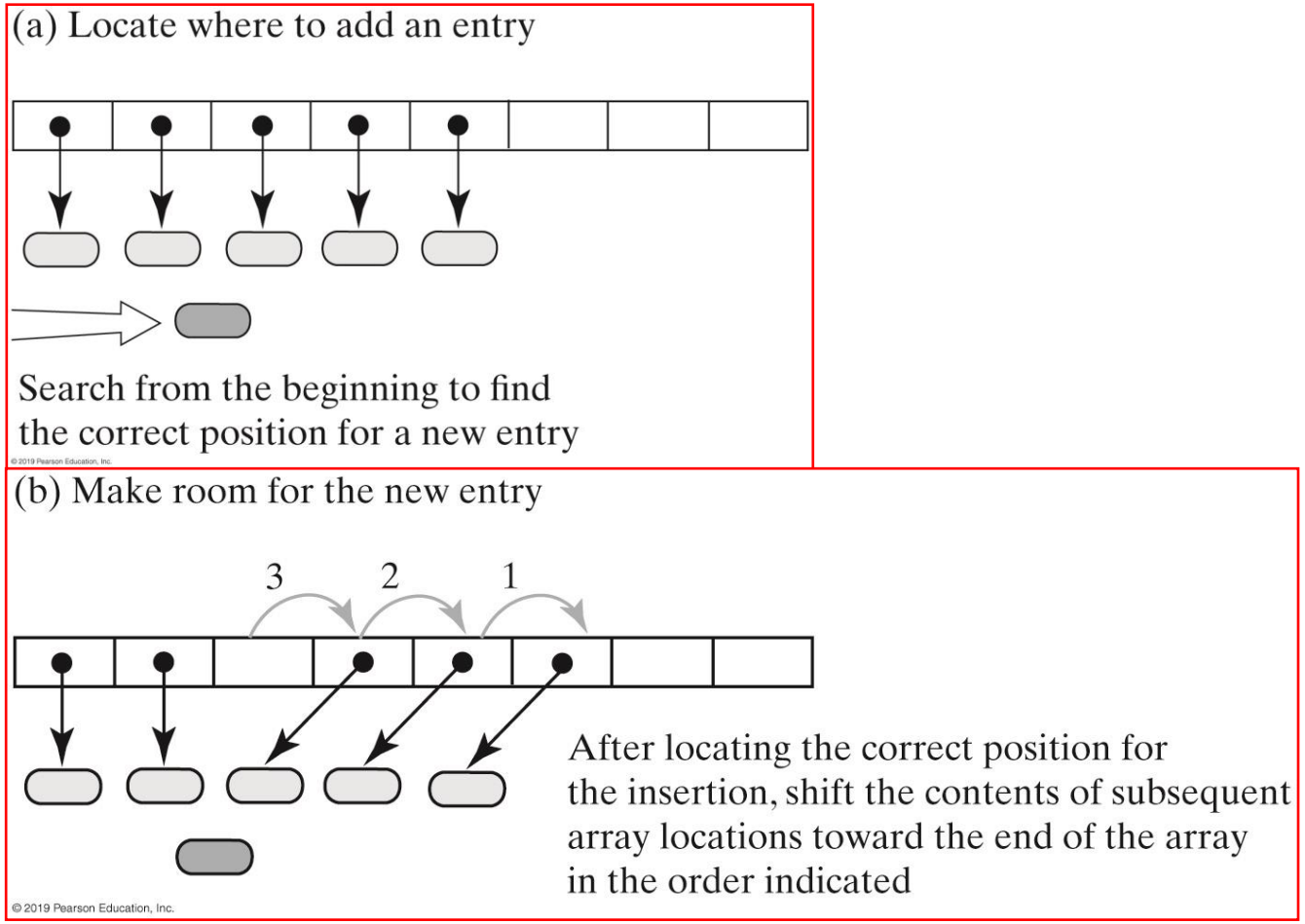
- Adding an entry (*entries are sorted*)



ASDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Sorted Array-Based Implementations

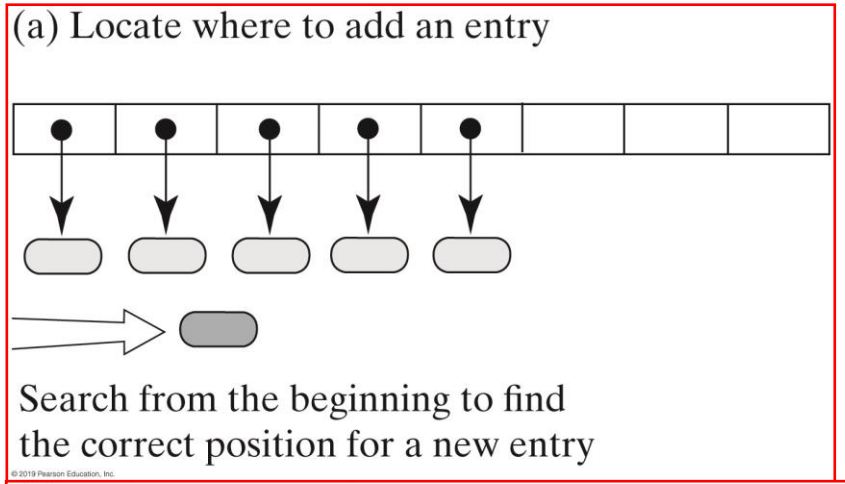
- Adding an entry (*entries are sorted*)

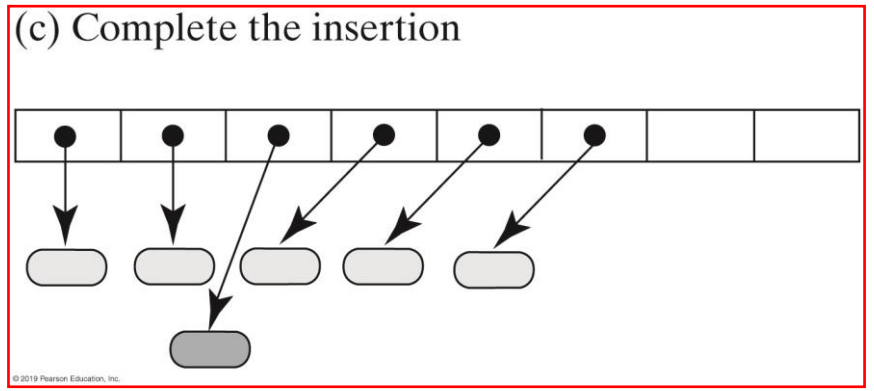


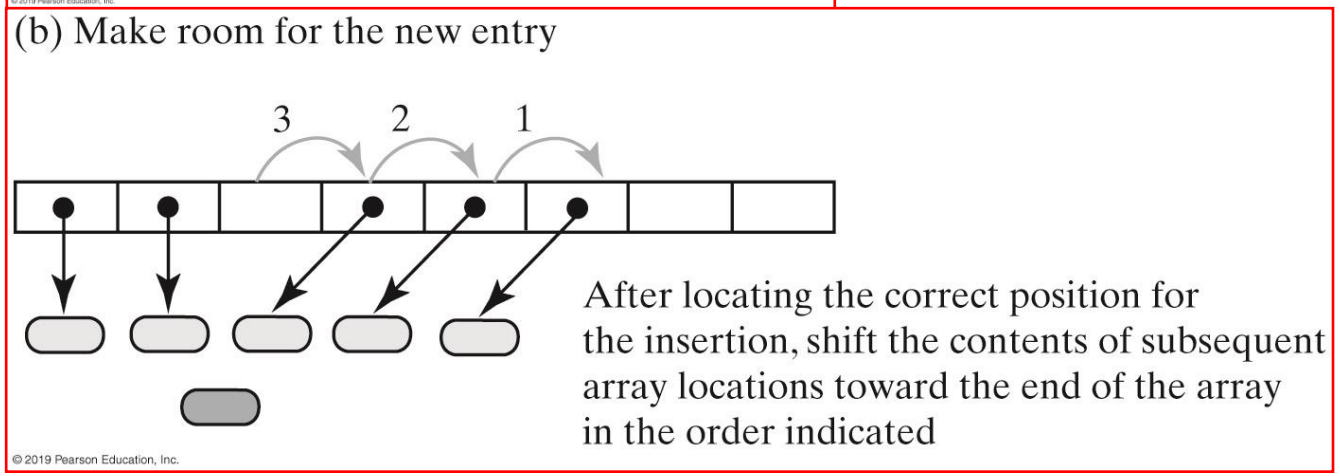
ASDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Sorted Array-Based Implementations

- Adding an entry (*entries are sorted*)

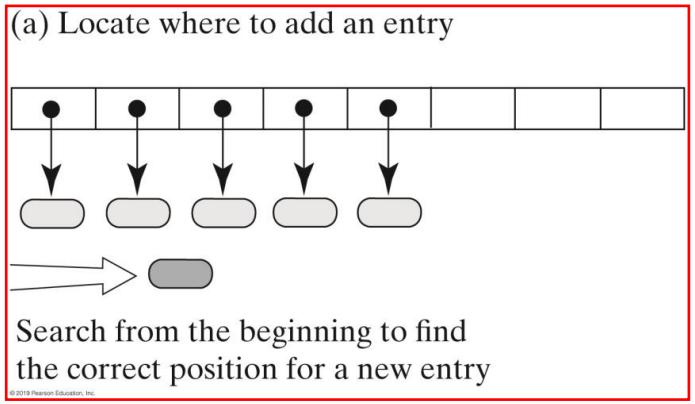


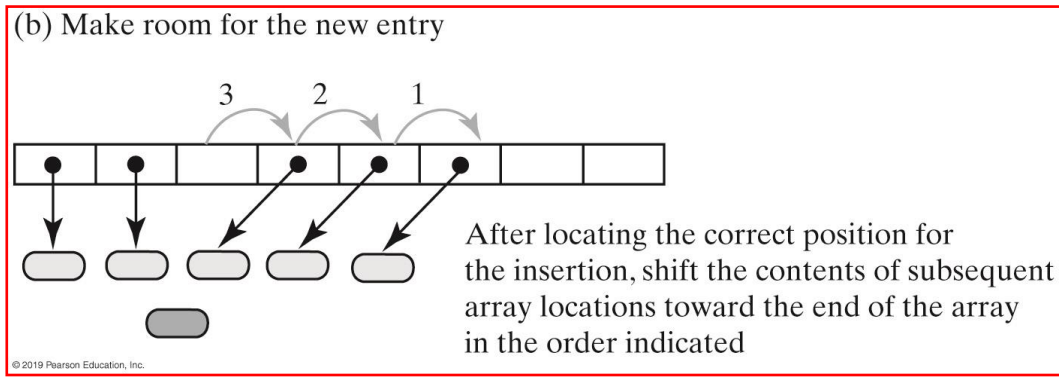


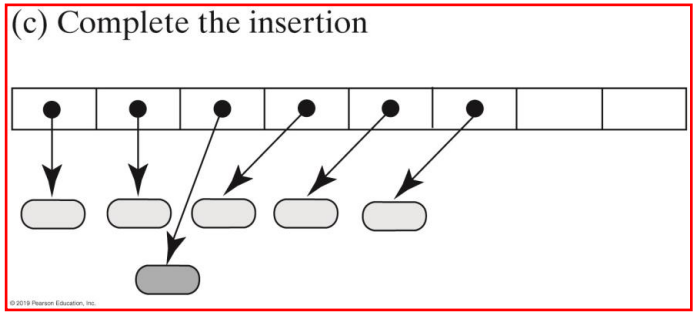


ASDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

```java
public V add(K key, V value)
{
      checkIntegrity();
      if ((key == null) || (value == null))
         throw new IllegalArgumentException("Cannot add null to a dictionary.");
      else
      {
         V result = null;
         int keyIndex = locateIndex_binarysearch(key);

         if ( (keyIndex < numberOfEntries) &&
               key.equals(dictionary[keyIndex].getKey()) )
         {
            // Key found, return and replace entry's value
            result = dictionary[keyIndex].getValue(); // Get old value
            dictionary[keyIndex].setValue(value); // Replace value
         }
         else // Key not found; add new entry to dictionary
         {
            makeRoom(keyIndex);
            dictionary[keyIndex] = new Entry<>(key, value);
            numberOfEntries++;
            ensureCapacity(); // Ensure enough room for next add
         } // end if

         return result;
      } // end if
} // end add
```



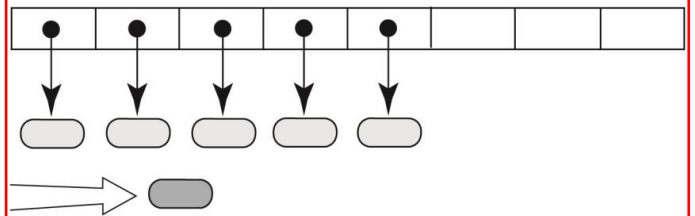




```java
public V add(K key, V value)
{
      checkIntegrity();
      if ((key == null) || (value == null))
         throw new IllegalArgumentException("Cannot add null to a
dictionary.");
      else
      {
         V result = null;
         int keyIndex = locateIndex_binarysearch(key);

         if ( (keyIndex < numberOfEntries) &&
               key.equals(dictionary[keyIndex].getKey()) )
         {
            // Key found, return and replace entry's value
            result = dictionary[keyIndex].getValue(); // Get old value
            dictionary[keyIndex].setValue(value); // Replace value
         }
         else // Key not found; add new entry to dictionary
         {
            makeRoom(keyIndex);
            dictionary[keyIndex] = new Entry<>(key, value);
            numberOfEntries++;
            ensureCapacity(); // Ensure enough room for next add
         } // end if

         return result;
      } // end if
} // end add
```

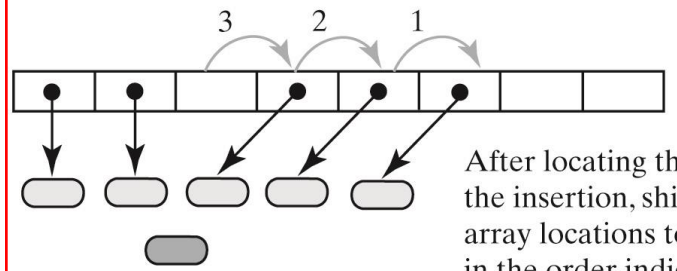
```java
// Makes room for a new entry at a given index by shifting
// array entries towards the end of the array.
// Precondition: keyIndex is valid; list length is old length.
private void makeRoom(int keyIndex)
{
      // Move each entry to next higher position beginning at
end of list
      // and continuing until item at newPosition is moved
      for (int index = numberOfEntries - 1; index >= keyIndex;
      index--)
      {
            dictionary[index+1] = dictionary[index]; // Shift
      right
      } // end for
} // end makeRoom
```

(a) Locate where to add an entry



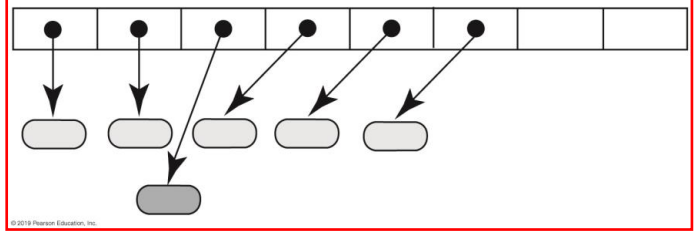
Search from the beginning to find the correct position for a new entry

(b) Make room for the new entry



After locating the correct position for the insertion, shift the contents of subsequent array locations toward the end of the array in the order indicated

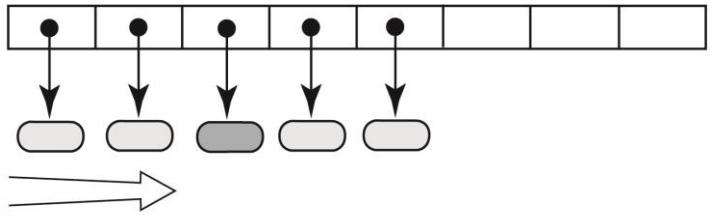
(c) Complete the insertion



Sorted Array-Based Implementations

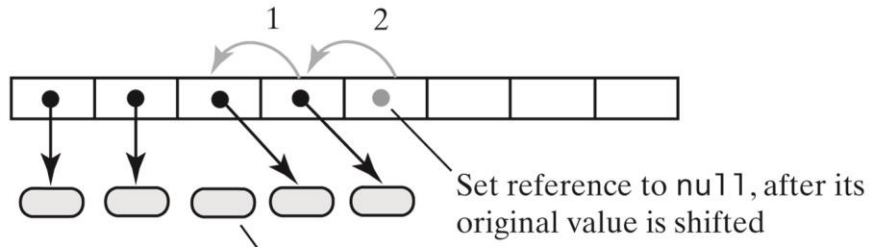
- Removing an entry (*entries are sorted*)

(a) Locate entry to remove



Search from the beginning to find the entry to remove

(b) Shift entries toward the one to remove

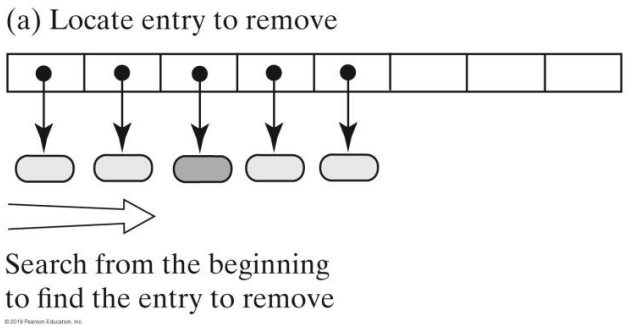


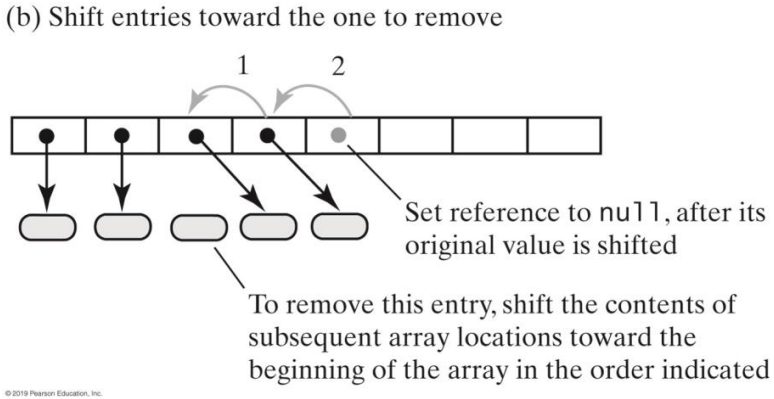
ASDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void **+remove(key : K) : V** +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Sorted Array-Based Implementations

- Removing an entry (*entries are sorted*)

```
public V remove(K key)
{
   checkIntegrity();
   V result = null;
   int keyIndex = locateIndex_binarysearch(key);

   if ( (keyIndex < numberOfEntries) &&
      key.equals(dictionary[keyIndex].getKey()) )
   {
         // Key found; remove entry and return its value
         result = dictionary[keyIndex].getValue();
         removeArrayEntry(keyIndex);
         numberOfEntries--;
   } // end if
   // Else result is null

   return result;
} // end remove
```





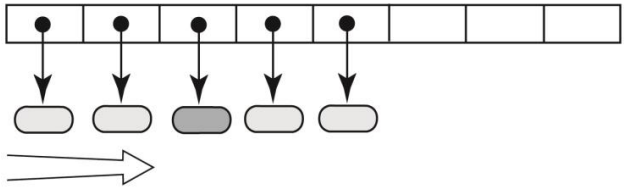
Sorted Array-Based Implementations

- Removing an entry (*entries are sorted*)

```
public V remove(K key)
{
   checkIntegrity();
   V result = null;
   int keyIndex = locateIndex_binarysearch(key);

   if ( (keyIndex < numberOfEntries) &&
      key.equals(dictionary[keyIndex].getKey()) )
   {
         // Key found; remove entry and return its value
         result = dictionary[keyIndex].getValue();
         removeArrayEntry(keyIndex);
         numberOfEntries--;
   } // end if
   // Else result is null

   return result;
} // end remove
```

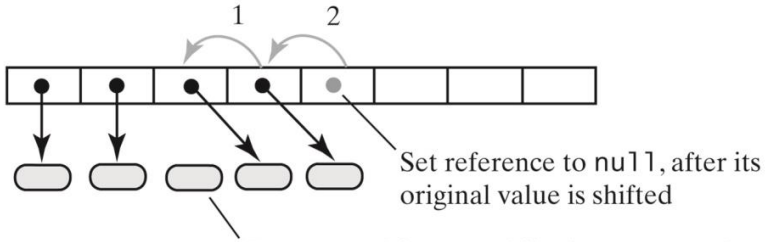
```
// Removes an entry at a given index by shifting array
// entries toward the entry to be removed.
private void removeArrayEntry(int keyIndex)
{
   for (int fromIndex = keyIndex+1; fromIndex < numberOfEntries; fromIndex++)
   {
      dictionary[fromIndex-1] = dictionary[fromIndex]; // Shift left
   } // end for
   dictionary[numberOfEntries - 1] = null;
}  // end removeArrayEntry
```

(a) Locate entry to remove



Search from the beginning to find the entry to remove

(b) Shift entries toward the one to remove



Sorted Array-Based Implementations

- What is the **Big Oh** of each dictionary method in the worst case?
  - Addition: $O(n)$
  - Removal: $O(n)$
  - Retrieval: $O(\log n)$

AUDictionary
-dictionary: Entry<K, V>[] -numberOfEntries: int -integrityOK: Boolean -DEFAULT_CAPACITY: int -MAX_CAPACITY: int
+add(key : K, value : V) : void **+remove(key : K) : V** **+getValue(key : K) : V** +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Implementations of a Dictionary

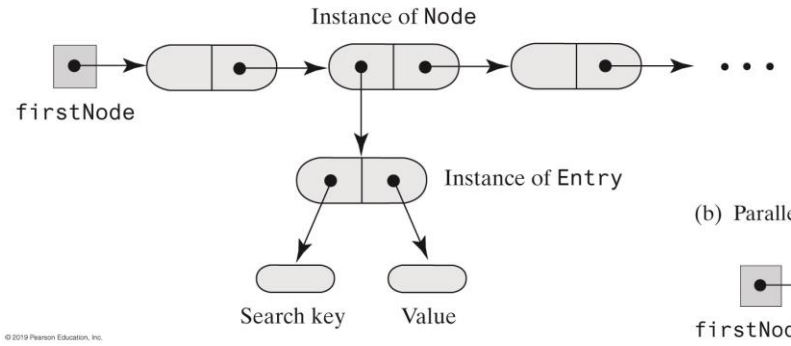
- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary
- **Linked Implementations**
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary
- Vector-Based Implementations

L(U/S)Dictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

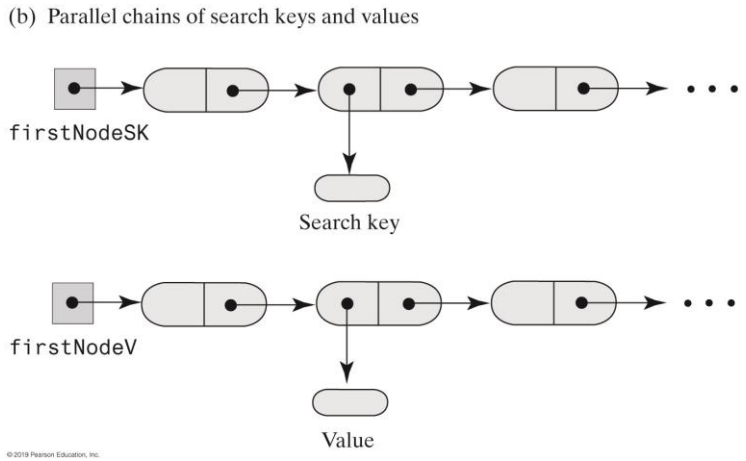
Representing the Entries in a Dictionary

- Three possible ways to use arrays to represent the entries in a dictionary

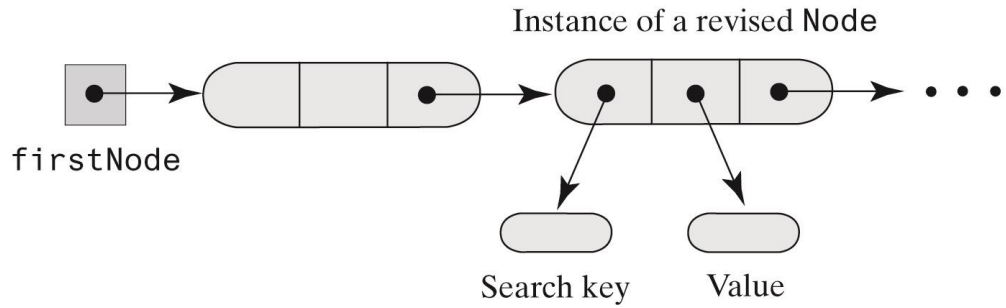
(a) A chain of nodes that each reference an entry object



(b) Parallel chains of search keys and values



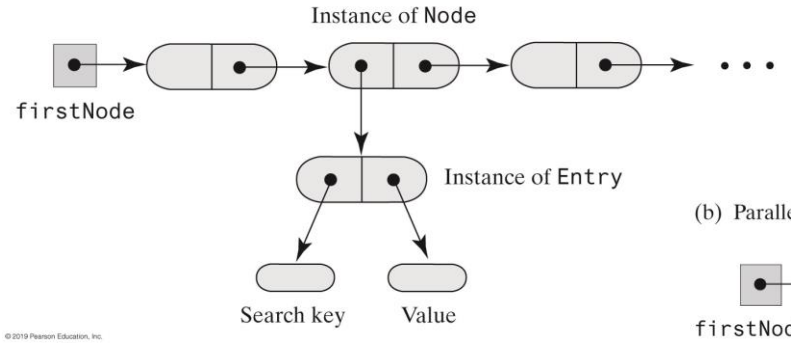
(c) A chain of nodes that each reference a search key and a value



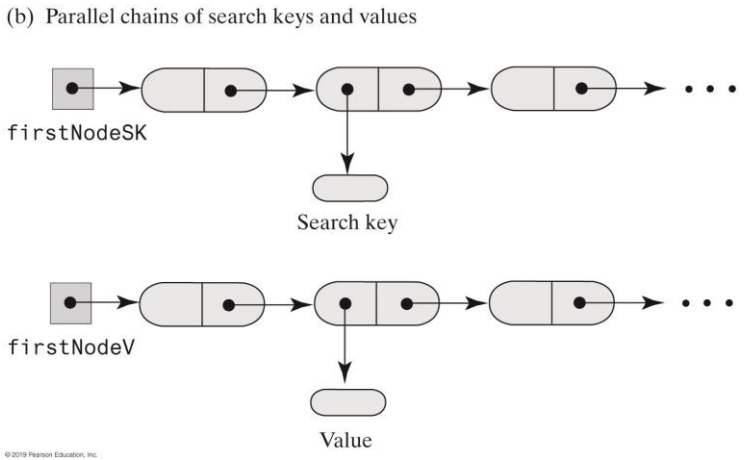
Representing the Entries in a Dictionary

- Three possible ways to use arrays to represent the entries in a dictionary

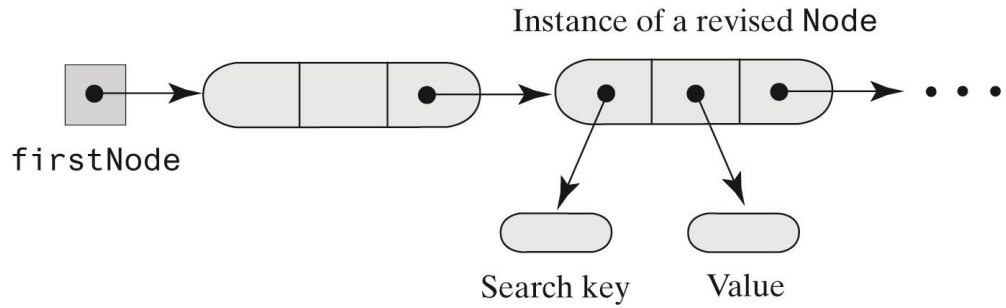
(a) A chain of nodes that each reference an entry object



(b) Parallel chains of search keys and values



(c) A chain of nodes that each reference a search key and a value



Implementations of a Dictionary

- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary
- **Linked Implementations**
  - **An Unsorted Linked Dictionary**
  - A Sorted Linked Dictionary
- Vector-Based Implementations

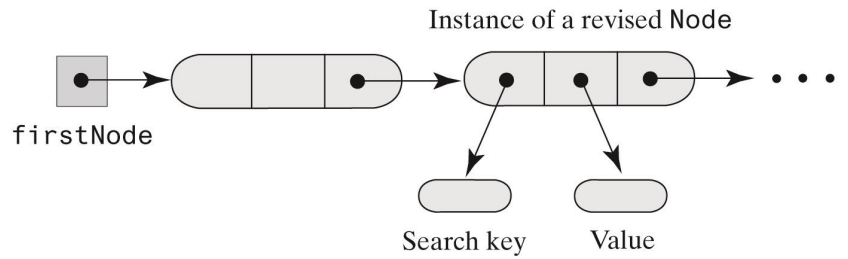
L(U/S)Dictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Implementations of a Dictionary

```
public class UnsortedLinkedDictionary<K, V> implements DictionaryInterface<K, V>
{
      private Node firstNode;   // Reference to first node of chain
      private int  numberOfEntries;

      public UnsortedLinkedDictionary()
      {
            initializeDataFields();
      } // end default constructor

      /* < Implementations of methods in DictionaryInterface. > . . . */

      private class Node
      {
            private K key;
            private V value;
            private Node next;

            private Node(K searchKey, V dataValue)
            {
                  key = searchKey;
                  value = dataValue;
                  next = null;
            } // end constructor

            private Node(K searchKey, V dataValue, Node nextNode)
            {
                  key = searchKey;
                  value = dataValue;
                  next = nextNode;
            } // end constructor
      } // end Node
} // end UnsortedLinkedDictionary
```

LUDictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

(c) A chain of nodes that each reference a search key and a value

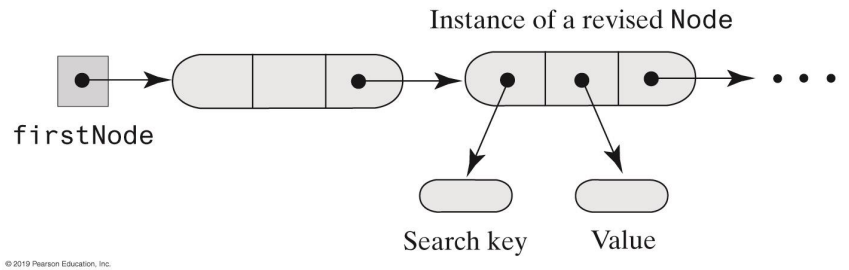


In-Class Exercises

- Write **add** method and **remove** method.

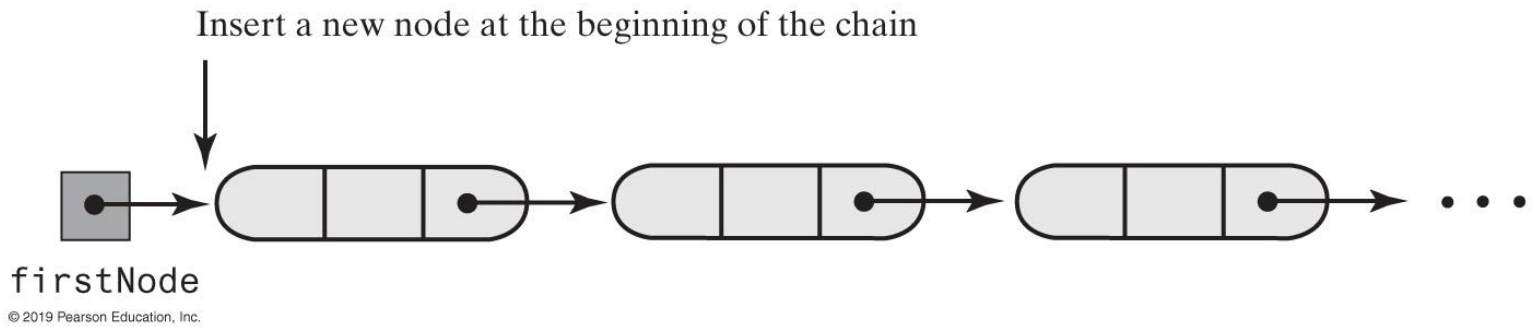
LUDictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

(c) A chain of nodes that each reference a search key and a value



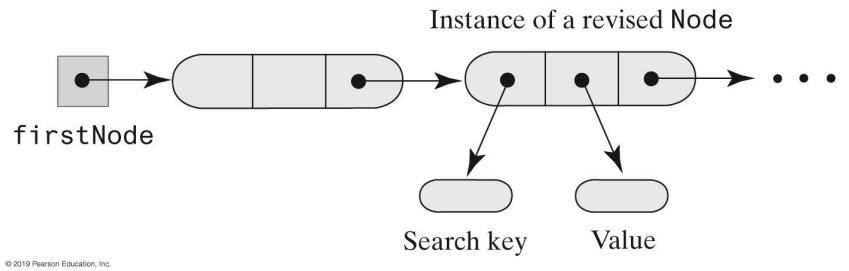
In-Class Exercises

- Write **add** method and **remove** method.



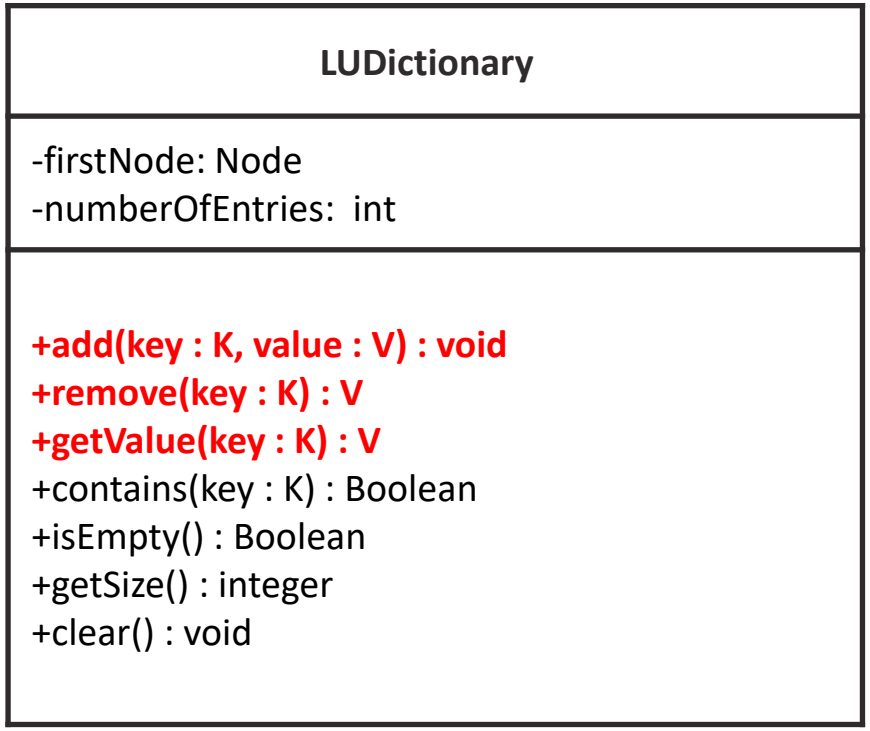
LUDictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void **+remove(key : K) : V** +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

(c) A chain of nodes that each reference a search key and a value



Unsorted Linked Implementations

- What is the **Big Oh** of each dictionary method in the worst case?
  - Addition: $O(n)$
  - Removal: $O(n)$
  - Retrieval: $O(n)$



Implementations of a Dictionary

- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary
- **Linked Implementations**
  - An Unsorted Linked Dictionary
  - **A Sorted Linked Dictionary**
- Vector-Based Implementations

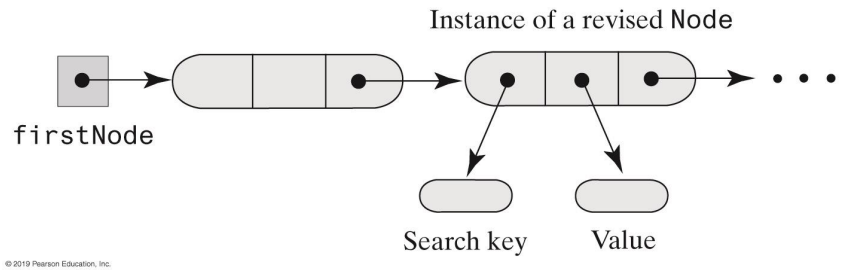
L(U/S)Dictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

In-Class Exercise

- Write and define your own ***Sorted Linked Dictionary***

LSDictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void +remove(key : K) : V +getValue(key : K) : V +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

(c) A chain of nodes that each reference a search key and a value



Sorted Linked Implementations

- What is the **Big Oh** of each dictionary method in the worst case?
  - Addition: $O(n)$
  - Removal: $O(n)$
  - Retrieval: $O(n)$

LSDictionary
-firstNode: Node -numberOfEntries: int
+add(key : K, value : V) : void **+remove(key : K) : V** **+getValue(key : K) : V** +contains(key : K) : Boolean +isEmpty() : Boolean +getSize() : integer +clear() : void

Summary

- Dictionaries
- Implementations of a Dictionary

What I Want You to Do

- Review class slides
- Review Chapters 20 and 21

- Next Topics
  - Hashing
  - Hashing as a Dictionary Implementation