[Java] `Map<KeyType, ValueType>` Interface (selected methods)

The classes that implement this interface are: **`TreeMap`** and **`HashMap`**.

`ValueType put(key, value)`
: Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced by the specified value. Returns the previous value associated with specified key, or `null` if there was no mapping for key.

`ValueType get(key)`
: Returns the value to which this map maps the specified key or `null` if the map contains no mapping for this key.

`boolean containsKey(key)`
: Returns `true` iff the map contains a mapping for the specified key.

`ValueType remove(key)`
: Removes the mapping for this key from this map if it is present, otherwise returns `null`.

`int size()`	Number of key-value mappings in this map.
`boolean isEmpty()`	Returns true if this map contains no key-value mappings.

`Set<Map.Entry<KeyType,ValueType>> entrySet()`
: Returns a set view of the entries contained in this map.

`Set<KeyType> keySet()`
: Returns a set view of the keys contained in this map.

[Java] `Map.Entry<KeyType, ValueType>` Interface

`KeyType getKey()`	Return the key of the entry
`ValueType getValue()`	Return the value of the entry
`void setValue(newVal)`	Replace the current value with `newVal`

[Java] `Iterator<ElmtType>` Interface (selected methods)

Some classes that implement this interface are: **`Scanner`**, **`ListIterator`**:

`boolean hasNext()`
: Returns `true` iff the iteration has more elements.

`ElmtType next()`
: Returns the next element in the iteration. Each successive call returns a different element in the underlying collection. For `Scanner` the `ElmtType` is always `String`.

[Java] `Collection<ElmtType>` Interface (selected methods)

Some classes that implement this interface are: **`ArrayList`**, **`LinkedList`**, **`TreeSet`**, and **`HashSet`**.

`boolean contains(elmt)`	Returns true iff `elmt` is in this collection
`int size()`	Returns number of elements in this collection
`boolean add(elmt)`	Ensures that `elmt` is in this collection. Returns true iff this collection changed as a result of this call
`boolean remove(elmt)`	Removes an instance of `elmt` from this collection. Returns true iff this collection changed as a result of this call
`boolean isEmpty()`	Returns true iff this collection contains no elements.
`Iterator<ElmtType> iterator()`	Returns an iterator over the elements in this collection.

[Java] `Concord` class (selected methods)

The `Concordance` class (from Lab 10) can compute the number of occurrences of all words from one or more Scanners. Converts a word to a standard form by stripping surrounding punctuation and converting it to all lower case before counting it.

`Concord()`
Creates an empty concordance

`void addData(Scanner in)`
Add data from `Scanner` to concordance. `in` will be at the end of its data after this operation.

`void print(PrintStream out)`
Write concordance data to out. Format is one entry per line: `word number`
where `number` is the number of occurrences of that word.

[Java] Select `File` and `Scanner` constructors

`File(String pathname)` Creates `File` object from given pathname. Does not open any file on disk.

`Scanner(File source) throws FileNotFoundException`
Constructs a new `Scanner` that produces values scanned from the specified file.
`FileNotFoundException` (a checked exception) is thrown if source file is not found on disk.

[C++] `Node` type and `ListType`

```
struct Node {
  int data;
  Node * next;

  Node(int item);  // create a Node with the given value and NULL next field

  // create a Node with the given data value (item) and next field (n)
  Node(int item, Node * n);

};

typedef Node * ListType;
```

[C++] selected C string functions

`int strlen( const char * str );`
Returns the length of the C string *str*. e.g., `strlen("foo")` returns `3`

`char * strcpy( char * destination, const char * source );`
Copies the C string pointed by *source* into the array pointed by *destination*, including the terminating null character (and stopping at that point).

`char * strcat( char * destination, const char * source );`
Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination.