Note: much of the documentation here is described in terms of interfaces, mostly to save space: in this way we don't have to describe every method for every class. Each "interface" section lists the classes we have learned about that implement the given interface. Also, it's all **Java**, except where C++ is specified.

`Stack<ElmtType>` class

`boolean empty()` tests if this stack is empty

`ElmtType peek()` looks at the top object of the stack (stack unchanged)

`ElmtType pop()` removes top element of the stack and return it

`ElmtType push(ElmtType item)` pushes an item onto the top of the stack

`Collection<ElmtType>` Interface

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

Selected methods:

`boolean contains(elmt)` Returns true iff this element is in this collection

`int size()` Returns number of elements in this collection

`boolean add(elmt)`
Ensures that element is in this collection.
Returns true iff this collection changed as a result of this call

`boolean remove(elmt)`
Removes an instance of this element 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.

`Iterator<ElmtType>` Interface

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

Selected methods:

`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`.

[More other side]

`Map<KeyType, ValueType>` Interface:

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

Selected methods:

`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.

`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.

Miscellaneous

Some other stuff you may have used before and forgotten.

Suppose you have an initialized `String` variable called `s`:

`new Scanner(s)`

constructs a new `Scanner` that produces values scanned from the specified string.

`s.trim()`

removes leading and trailing whitespace from a string (returns a copy with the changes)

`s.split(regex)`

splits this string around matches of given regular expression and returns the pieces in an array of strings. regex is type String.

`Node` type and `ListType` (this is the only part of the code handout with C++ code):

```
struct Node {
  int data;
  Node * next;
  Node() { data = 0; next = NULL; }
  Node(int d) { data = d; next = NULL; }
  Node(int d, Node * n) { data = d; next = n; }
};

typedef Node * ListType;
```

[More other side]