(C++) `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;
```

(Java) Reminder of syntax to implement an interface (using `Comparator<Type>` interface)

```
class StringLengthComparator implements Comparator<String> {

   public int compare(String s1, String s2) {

      return s2.length() - s1.length();

   }

}
```

(Java) Selected methods of `Point` class:

`Point p = new Point(x, y)` create a point with coordinates (x,y)

`p.getX()` `p.getY()` returns x and y coordinates of `p`, respectively.

`p.translate(dx, dy)`
Changes `x` and `y` values of `p` by `dx` and `dy`, respectively. I.e., if `p` had coordinates `(x, y)`, its new value is a point with coordinates `(x+dx, y+dy)`

`p.setLocation(x, y)` Moves point to location (x,y).

`p.distance(p2)` Returns the distance (a `double`) from `p` to a specified `Point`.

`p.distance(double px, double py)` Returns the distance (a `double`) from `Point p` to a specified point.

`p.distanceSq(p2)` and `p.distanceSq(double px, double py)`
Like distance methods above, but returns the square of the distance instead.

(Java) `Collections` class (selected methods)

The `Collections` class contains static methods that operate on collections. Note: `ArrayList` and `LinkedList` both implement the `List` interface used below.

`static void sort(List<ElmtType> list)`
Sorts the list into ascending order according to the natural ordering of its elements (i.e., using `compareTo`).

`static int binarySearch(List<ElmtType> list, ElmtType target)`
Searches the list for target using the binary search algorithm. The list elements must implement the `Comparable` interface. The list must be sorted into ascending order. Returns the position of the target if it's in the list; otherwise returns *–index - 1*, where *index* is the position where the element may be inserted.

[continued other side]

(Java) `Map<KeyType, ValueType>` Interface:

The 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.

`Set<Map.Entry<KeyType,ValueType>> entrySet()`
Returns a set view of the entries 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) `Collection<ElmtType>` Interface

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

`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) `Iterator<ElmtType>` Interface

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

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`.

`void remove()`
Removes from the underlying collection the last element returned by the iterator. (`Scanner` does not implement this optional method.)