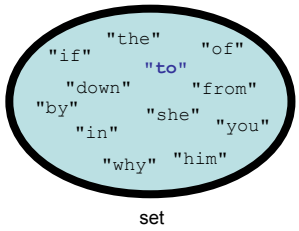
CSE 373 Section Handout #2
Syntax Reference

`Collection<E>` Methods

(methods found in sets, lists, and other collections)



set

`add(`**value**`)`	adds the given value to the set
`addAll(`**collection**`)`	add all elements from given collection to this one
`clear()`	removes all elements of the set
`contains(`**value**`)`	`true` if the given value is found in the set
`isEmpty()`	returns `true` if the set's size is 0
`remove(`**value**`)`	removes the given value from the set
`size()`	returns the number of elements in the set
`toString()`	text representation such as `"[1, 2, 3]"`
`addAll(`**collection**`)`	adds all elements from the given collection to the set
`containsAll(`**collection**`)`	returns `true` if set contains every element from given collection
`equals(`**collection**`)`	returns `true` if the given other set contains the same elements
`iterator()`	returns an object used to examine the contents of the set
`removeAll(`**collection**`)`	removes any elements found in the given collection from this set
`retainAll(`**collection**`)`	removes any elements *not* found in the given collection from this set
`toArray()`	returns an array of the elements in this set
`toString()`	returns a string representation of the set such as `"[10, -2, 43]"`

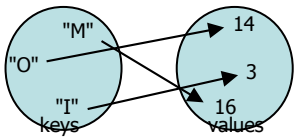
List<E> Methods (10.1)

(Java's implementation of a list of any type)

`add(`**index**`, `**value**`)`	inserts given value at given index, shifting subsequent values right
`addAll(`**index**`,`**collection**`)`	adds all elements from given collection to this list at the given index
`get(`**index**`)`	returns the value at given index
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`listIterator()`	returns an object used to examine the contents of the list
`lastIndexOf(`**value**`)`	returns last index where given value is found in list (-1 if not found)
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`subList(`**from**`, `**to**`)`	sub-portion of the list indexes **from** (inclusive) and **to** (exclusive)

`Map<K, V>` Methods

(A fast mapping between a set of keys and a set of values)



`put(`**key**`, `**value**`)`	adds a mapping from the given key to the given value
`get(`**key**`)`	the value mapped to the given key (`null` if none)
`containsKey(`**key**`)`	`true` if the map contains a mapping for the given key
`remove(`**key**`)`	removes any existing mapping for the given key
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value pairs in the map
`isEmpty()`	returns `true` if the map's size is 0
`toString()`	returns a string such as `"{O=14, M=16, I=3}"`
`keySet()`	returns a `Set` of all keys in the map
`values()`	returns a `Collection` of all values in the map
`putAll(`**map**`)`	adds all key/value pairs from the given map to this map
`equals(`**map**`)`	returns `true` if given map has same key/value pairs as this one

CSE 373 Section Handout #2

Choosing a Collection

1. Standard Collections

What is a good Java collection to use to represent each of the following? Justify your answer.

a) the lines of text in a file (some lines might be duplicates of other lines)
b) your Buddy List on a chat program (names of your buddies' accounts)
c) an Undo feature in an editor program (enables you to undo the most recent action(s) performed)
d) the waiting list to get a table at a popular restaurant, which should be accommodated in a first-come, first-served order
e) a telephone book where you can look up someone's phone number by their name
f) the purchase orders received by Amazon.com that need to be processed and shipped out, generally in the same order that the orders were placed by the customers, but with "rush processing" for orders that pay extra for overnight shipping
g) a listing of prime numbers, in ascending order

2. Compound Collections

What is a good *compound* Java collection to use to represent each of the following? Justify your answer.

a) the lines of text in a file, broken up into words (but remembering where the breaks are between lines)
b) ALL users' buddy lists in a chat server (so we can quickly look up a user's buddy list by their user name)
c) an Urban Dictionary of phrases, where each phrase can have one or more definitions
d) the waiting list to get a table at a popular restaurant, where different tables have different sizes, and customers want to wait for a specific table that is exactly the right size for them
e) a collection of word lengths, where it is easy/efficient to answer the question, "What are all words in the English dictionary of length N?"
f) data about students' grades, where given a course name and a student's name, you can quickly discover that student's grade (0.0 to 4.0) in that course
g) a collection of 3-dimensional points of the form (x, y, z)
h) an index in a textbook, where you want to know what page number(s) each specific term appears on

Lists, Sets, and Maps

3. `removeTens`

Write a method named `removeTens` that accepts a `List` of integers as a parameter and that removes all of the integers that are multiples of 10 from the list. For example, if the list `[50, 2, 580, 60, 19, -30]` is passed, your code should change the list to be `[2, 19, -4]`.

4. `contains3`

Write a method named `contains3` that accepts a `List` of strings as a parameter and returns `true` if any single string occurs at least 3 times in the list, else `false`. You may use one collection as auxiliary storage.

5. `isUnique`

Write a method named `isUnique` that accepts a `Map` from strings to strings as a parameter and returns `true` if no two keys map to the same value (and `false` if any two or more keys do map to the same value). For example, calling your method on the following map would return `true`:

```
{Marty=Stepp, Stuart=Reges, Jessica=Miller, Amanda=Camp, Hal=Perkins}
```

Calling it on the following map would return `false`, because of two mappings for `Perkins` and `Reges`:

```
{Kendrick=Perkins, Stuart=Reges, Jessica=Miller, Bruce=Reges, Hal=Perkins}
```

CSE 373 Section Handout #2

Complexity and Efficiency

6. Binary Search Complexity

Approximately what is the maximum number of elements that must be examined to perform a binary search on an array of size *N*? What does this imply about the complexity class of the binary search algorithm, and how does this compare to the complexity class of the standard sequential search algorithm? On average, how many elements must each algorithm examine to search an array containing 1,000,000 elements?

7. Complexity Classes

To which complexity class does each of the following algorithms belong?
Consider *N* to be the length or size of the array or collection passed to the method.

a)
```
public static int[] stretch(int[] list) {
    int[] result = new int[2 * list.length];
    for (int i = 0; i < list.length; i++) {
        result[2 * i] = list[i] / 2 + list[i] % 2;
        result[2 * i + 1] = list[i] / 2;
    }
    return result;
}
```

b)
```
public static int maxDiff(int[] list) {
    int diff = 0;
    for (int i = 0; i < list.length; i++) {
        for (int j = i + 1; j < list.length; j++) {
            diff = Math.max(diff, Math.abs(i - j));
        }
    }
    return diff;
}
```

c)
```
public static void switchPairs(List<String> list) {
    for (int i = 0; i < list.size() - 1; i += 2) {
        String first = list.remove(i);
        list.add(i + 1, first);
    }
}
```

d)
```
public static void switchPairs(List<String> list) {
    for (int i = 0; i < list.size() - 1; i += 2) {
        String first = list.get(i);
        list.set(i, list.get(i + 1));
        list.set(i + 1, first);
    }
}
```

e)
```
public static void collapse(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty())
        q.add(s.pop());
    while (!q.isEmpty())
        s.push(q.remove());
    while (!s.isEmpty())
        q.add(s.pop());
    while (q.size() > 1)
        s.push(q.remove() + q.remove());
    if (!q.isEmpty())
        s.push(q.remove());
}
```

CSE 373 Section Handout #2
Solutions

1. There is often more than one acceptable answer to questions like these. Here are our choices:
   a) `List<String>`
   b) `TreeSet<String>`
   c) `Stack`
   d) `Queue<String>`
   e) `Map<String, String>`
   f) `PriorityQueue`
   g) `TreeSet<Integer>` or `List<Integer>`

2. There is often more than one acceptable answer to questions like these. Here are our choices:
   a) `List<List<String>>`
   b) `Map<String, Set<String>>`
   c) `Map<String, List<String>>`
   d) `Map<Integer, Queue<String>>`
   e) `Map<Integer, Set<String>>` or `List<Set<String>>` where index `i` stores strings of length `i`
   f) `Map<String, Map<String, Double>>`
   g) `List<int[]>` or `List<Point3D>`
   h) `Map<String, Set<Integer>>`

3. Three possible solutions are shown.

```
public void removeTens(List<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) % 10 == 0) {
            list.remove(i);
            i--;
        }
    }
}

public void removeTens(List<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        if (list.get(i) % 10 == 0) {
            list.remove(i);
        }
    }
}

public void removeTens(List<Integer> list) {
    Iterator<Integer> itr = list.iterator();
    while (itr.hasNext()) {
        int n = itr.next();
        if (n % 10 == 0) {
            itr.remove();
        }
    }
}
```

CSE 373 Section Handout #2

4.

```
public static boolean contains3(List<String> list) {
    Map<String, Integer> counts = new HashMap<String, Integer>();
    for (String value : list) {
        if (counts.containsKey(value)) {
            int count = counts.get(value);
            count++;
            counts.put(value, count);
            if (count >= 3) {
                return true;
            }
        } else {
            counts.put(value, 1);
        }
    }

    return false;
}
```

5. Two possible solutions are shown.

```
public static boolean isUnique(Map<String, String> map) {
    Set<String> values = new HashSet();
    for (String value : map.values()) {
        if (values.contains(value)) {
            return false;   // duplicate
        } else {
            values.add(value);
        }
    }
    return true;
}

public static boolean isUnique(Map<String, String> map) {
    return new HashSet(map.values()).size() == map.values().size();
}
```

6.

The number of elements that must be examined is approximately equal to the number of times the array must be divided in half until the remaining length is 1 element; if we call this quantity x, we would say that $2^x = N$ or $x = \log_2 N$. This implies that binary search is in the logarithmic complexity class, O(log N). The standard sequential search algorithm is in the linear complexity class, O(N). To examine an array with one million elements, sequential search might have to examine all elements to find the value, and examines about half of them on average when the element is found. Binary search would have to examine at most $\log_2 1000000 = 20$ elements. (Much faster!)

7.

a) O(N)
b) O(N^2) (even though the inner loop goes an average of N/2 times, we ignore constants like ½)
c) O(N^2) (the loop repeats roughly N/2 times, and the operations in the loop (`remove`, `add`) are O(N))
d) O(N) for an `ArrayList`; the loop repeats ~N/2 times, the operations in the loop (`get`, `set`) are O(1)). O(N^2) for a `LinkedList`, because operations get and set are O(N).
e) O(N) (there are many loops, but each runs ~ N times and none are nested, so we add their runtimes)