Here are some Abstract Data Types that you have seen in class so far. We want to get some practice with using them. First, let's review the following ADTs.

List

A **list** is an ordered sequence of items. It is like an array, but can have variable length/size.

```
interface List<T> {
  void    add(T item);
  void insert(T item, int position); // insert item at this position
  T       get(int position);
  int    size();
}
```

Note: `List` is an interface; you cannot directly make a List. However, ArrayLists are Lists, and you can make them with: `List myList = new ArrayList<T>()`, where `T` is the item type.

Set

A **set** is an unordered collection of non-duplicate items.

```
interface Set<T> {
  void          add(T item);  // If item is already in the set, nothing changes.
  boolean contains(T item);
  List<T>    items();         // return a List of all items in some arbitrary order
  int          size();
  void       remove(T item);
}
```

Note: `Set` is an interface; you cannot directly make a Set. However, HashSets are Sets, and you can make them with: `Set mySet = new HashSet<T>()`, where `T` is the item type.

Map

A **map** associates or "maps" keys to values. Python has this concept too! In Python they're called dictionaries. You can also think of a map as a set of <key, value> pairs, except that keys cannot be duplicated, and that looking up an value by key is fast (constant time).

```
interface Map<K, V> {
  void    put(K key, V value); // put key into the map and associates it with
                               // value, replacing old value if it exists
  boolean containsKey(K key);
  List<K> keys();              // returns a List of all keys in some arbitrary order
}
```

Note: `Map` is an interface; you cannot directly make a Map. However, HashMaps are Maps, and you can make them with: `Map myMap = new HashMap<K, V>()`, where `K` is the key type and `V` is the value type.

Now let's write some methods!

1. Does arr have duplicates?

```
public static boolean findDups(int[] arr) {

   Set<Integer> seen = new HashSet<Integer>();
   for (int item : arr) {
      if (seen.contains(item)) return true;
      seen.add(item);
   }

   return false;

}
```

2. Do any two elements in the array sum up to n?

Hint: You don't need to check all combinations.

Solution using an an enhanced for-loop.

```
public static boolean sumUp(int n, int[] arr) {

   Set<Integer> seen = new HashSet<Integer>();
   for (int item : arr) {
      if (seen.contains(n - item)) return true;
      seen.add(item);
   }
   return false;

}
```

Solution using a traditional for-loop. Solutions for later problems will only have the enhanced for-loop solution for brevity.

```
public static boolean sumUp(int n, int[] arr) {

   Set<Integer> seen = new HashSet<Integer>();
   for (int i = 0; i < arr.length; i++) {
      int item = arr[i];
      if (seen.contains(n - item)) return true;
      seen.add(item);
   }
   return false;

}
```

3. Missing number

arr contains all the numbers from `0` to `n` for some `n` except some number `k`.
Find `k`. *Don't worry about what happens if the precondition is not met.*

Three different solutions are given here:

1. Using a `Set` to store seen numbers
2. Using an array to store whether each number has been seen
3. Subtracting the sum of the seen numbers from the expected sum

```
public static int missingNumber(int[] arr) {

   1. Set<Integer> seen = new HashSet<Integer>();
   2. boolean[] seen = new boolean[arr.length+1];
   3. sum = 0;

   for (int item : arr) {
      1. seen.add(item);
      2. seen[item] = true;
      3. sum += item;
   }

   1. for (int i = 0; i < arr.length + 1; i++) {
          if (!seen.contains(i)) return i;
      }
   2. for (int i = 0; i < seen.length; i++) {
         if (!seen[i]) return i;
      }
   3. // Add up everything from 0 to arr.length.
      // Can't math? www.wolframalpha.com/input/?i=sum+of+x+from+0+to+n
      int expectedSum = (arr.length * (arr.length + 1))/2;
      return expectedSum - sum;

}
```

4. Is s1 a permutation of s2?

To review: The permutations of `cat` are: `cat`, `cta`, `act`, `atc`, `tca`, `tac`.

Hint: Use a Map.

The strategy: Go through `s1` and count the number of occurences of each letter, keeping track in a counter map. Using the same counter, go though `s2` except count in reverse (subtract 1 for each occurrence). In the end the counter should be all zeroes.

```
public static boolean isPermutation(String s1, String s2) {

   // Mapping character to its count
   Map<Character, Integer> charCounts = new HashMap<Character, Integer>();

   for (int i = 0; i < s1.length(); i++) {
     char c = s1.charAt(i);
     int oldCount = 0;
     if (charCounts.containsKey(c)) oldCount = charCounts.get(c);
     charCounts.put(c, oldCount + 1);
   }

   for (int i = 0; i < s2.length(); i++) {
     char c = s2.charAt(i);
     int oldCount = 0;
     if (charCounts.containsKey(c)) oldCount = charCounts.get(c);
     charCounts.put(c, oldCount - 1);
   }

   for (char c : charCounts.keys()) {
      if (charCounts.get(c) != 0)) return false;
   }
   return true;

}
```

5. Finding duplicates within a range

Given an `int[] a` and a boundary range `k`, find if there are any duplicates that are within `k` indices of each other. Examples:

- `findDuplicatesWithinK([1,2,3,1,4,3], 3) -> [1,3]`
- `findDuplicatesWithinK([1,2,3,1,4,3], 2) -> []`

Hint: If you end up with a `Set` or `List` of duplicates, here's how you can convert it to an array:

```
your_set_or_list.toArray(new int[your_set_or_list.size()])
```

Note that this solution is based of the solution of problem 1 (`findDups`) above.

```
public static int[] findDuplicatesWithinK(int[] a, int k) {

   Set<Integer> seen = new HashSet<Integer>();
   Set<Integer> dups = new HashSet<Integer>();
   for (int i = 0; i < a.length; i++) {
      int item = a[i];
      if (seen.contains(item)) dups.add(item);
      seen.add(item);
      if (i-k >= 0) seen.remove(a[i-k]);
   }

   return dups.toArray(new int[dups.size()]);
   // Java's toArray methods are weird because you need to pass it
   // an empty array of the correct type. If you want, it's acceptable
   // to convert it to an array manually as well.

}
```