Sets

- As you discovered last time by looking up the Set API:
  - Almost everything has been done for you
- Union of two sets – Use:
  - `boolean addAll(Collection c)`
- Intersection of two sets – Use:
  - `boolean retainAll(Collection c)`
- Difference of two sets (A – B) – Use:
  - `boolean removeAll(Collection c)`

Collections: ArrayList vs Set

- **ArrayList**
  - directly access an item
  - keep items ordered (as entered)
  - Can have duplicates of items
  - What are operations on an ArrayList?
- **Sets**
  - Keeps items unordered
  - (Unless using TreeSet: sorted)
  - No duplicate items
  - Easily remove duplicates
  - What are operations on a Set?

Using Both ArrayList and Sets

- You may want to use a set to get rid of duplicates, then put the items in an ArrayList and sort them!
- **Problem:**
  - Often data comes in the form of an array
  - How do we go from array to ArrayList or TreeSet?
- **Problem:**
  - Often we are required to return an array
  - How do we go from a Collection such as an ArrayList or TreeSet to an array?
- Can do it the “hard” way with loops or iterators:
  - one item at a time
- OR:

Converting from array to Collection

- For arrays of objects (such as Strings) use the `asList` method in the `Arrays` class.
  - This returns a fixed-size list backed by the specified array
  - Pass this into the constructor of your ArrayList or set
- **Example**
  ```java
  String[] words = String[N];
  ...
  TreeSet<String> wordset = new TreeSet<String>(Arrays.asList(words));
  ```
Converting from Collection to array

- Collections such as ArrayLists and TreeSet have a toArray method
  - This returns an array
  - Syntax a bit awkward
- Example
  
  ```java
  TreeSet<String> wordset = new TreeSet<String>();
  ...
  String[] words =
    (String[]) wordset.toArray(new String[0]);
  or
  return (String[]) wordset.toArray(new String[0]);
  ```