1 Comparable

```
class Author implements Comparable<Author>{
    String firstName;
    String lastName;

    @Override
    public int compareTo(Author other){
        // compareTo should return < 0 if this is supposed to be
        // less than other, > 0 if this is supposed to be greater than
        // other and 0 if they are supposed to be equal
        int last = this.lastName.compareTo(other.lastName);
        return last == 0 ? this.firstName.compareTo(other.firstName) : last;
    }
}
```

2 Quickselect

```
public class QuickSelect {
    private static void swap(int[] arr, int left, int right) {
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
    }

    public static int partition(int[] arr, int left, int right, int pivot) {
        int pivotVal = arr[pivot];
        swap(arr, pivot, left);
        int low = left;
        int high = right+1;
        while(true) {
            while(arr[++low] <= pivotVal) {
                if( low == right) break;
            }
            while(arr[--high] > pivotVal) {
                if( high == left) break;
            }

            if (low < high) {
                swap (arr, low, high);
            } else {
                swap(arr, high, left);
                break;
            }
        }

        return high;
    }

    public static int select(int[] arr, int k) {
        int ans = k-1; // k - 1 because arrays are 0-indexed
        Random rand = new java.util.Random();
        int found = -1;
        // select a pivot
        int start = 0;
        int end = arr.length - 1;

        while (start <= end) {
            int pivot = rand.nextInt(end-start + 1) + start;
            int pivotIndex = partition(arr, start, end, pivot);
            if (pivotIndex == ans) {
                return found = arr[pivotIndex];
            } else if (pivotIndex < ans) {
                start = pivotIndex + 1;
            } else {
                end = pivotIndex - 1;
            }
        }

        return found;
    }
}
```

3 Collections.sort

`compare` returns `0` if the objects are equal. It returns a positive value if `obj1` is greater than `obj2`. Otherwise, a negative value is returned. Below snippet reversees the array.

```
ArrayList<Integer> stuff =  new ArrayList<>(Arrays.asList(1,2,3,4));
Collections.sort(stuff, new Comparator<Integer>() {
        @Override
        public int compare(Integer i1, Integer i2) {
            if (i1 < i2) {
                return 1;
            } else {
                return -1;
            }
        }
    });
```

4 Iterators

```
@Override
public Iterator<Type> iterator() {
    Iterator<Type> it = new Iterator<Type>() {

            private int currentIndex = 0;

            @Override
            public boolean hasNext() {
                return currentIndex < currentSize && arrayList[currentIndex] != null;
            }

            @Override
            public Type next() {
                return arrayList[currentIndex++];
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
    return it;
}
```

5 Interfaces

5.1 Stack

```
push(T): void
pop(): T
size(): T
```

```
isEmpty(): boolean
top(): T
```

5.2 Queue

```
offer(T): boolean
peek(): T (returns null if empty)
poll(): T (returns null if empty)
```

5.3 Set

```
add(T) : boolean
addAll(Coll): boolean
clear(): void
contains(T): boolean
containsAll(T): boolean
isEmpty(): boolean
remove(T): boolean
removeAll(Coll): boolean
size(): int
```