# Test 1 Review

## Compilation

- the vast majority of submissions compiled
- most common reasons for failure to compile
  - ran out of time
  - forgot to ensure that a return statement was reached
  - unable to throw an exception correctly

## How to throw an exception

- in Java, an exception is an object
  - therefore, you have to create a new exception object

```
public static char second(String s) {
  if (s.length() < 2) {
    throw new IllegalArgumentException("too short!");
  }
  return s.charAt(1);
}
```

## Return statements

- a method that returns a value must ensure that a value is always returned when the method completes
  - the following won't compile

```
public static char second(String s) {
  if (s.length() >= 2) {
    return s.charAt(1);
  }
  // forgot to throw exception here;
  // now you need a return statement
}
```

## Compilation against the tester

- what happens if your class does not compile against the tester?
  - your class does not provide the required API
  - we can't test your code
    - this includes all methods that might be correct

## Compilation against the tester

- how can your class compile, but fail to compile against the tester?
  - your class is not providing the required API
    - your class does not implement the correct interface
    - you are missing a public field
    - you have misspelled the name of a public field
    - you have specified an incorrect type for a public field
    - you are missing a public method
    - you have misspelled the name of a public method
    - you have the wrong parameter list in a public method
    - you have specified an incorrect return type for a public method

## Compilation against the tester

- it is easy to ensure that your class will compile against the tester
  1. copy the class header from the API
  2. copy the public field declarations from the API
  3. copy the public method headers from the API
  4. add any legal return statement to every method that returns a value
  5. add the required import statements

```java
import java.util.List;
import java.util.Set;

public class Test2C {
  public static char first(String s) {
    return 'a';
  }

  public static char second(String s) {
    return 'a';
  }

  public static String longest(Set<String> t) {
    return "";
  }

  public static String mostFrequent(List<String> t) {
    return "";
  }
}
```

## Compilation against the tester

- if you have trouble implementing a method, do not comment out the entire method
  - this will prevent the tester from compiling
- instead, comment out the method body
  - if the method returns a value, insert any valid return statement

```
public class Test2C {
  // 3 methods not shown

  /*
  public static String mostFrequent(List<String> t) {
    List<String> u = new List<String>(t);
    return u.get(0);
  }
  */

  public static String mostFrequent(List<String> t) {
    /*
    List<String> u = new List<String>(t);
    return u.get(0);
    */
    return "";
  }
}
```

Don't do this; the tester won't compile.

Do this instead; the tester compiles, and we can still see your attempt.

# Test2C

- Given a list of strings, find the string that occurs the most frequently in the list. If more than one string occurs most frequently, return the string that occurs first alphabetically and occurs most frequently.

- e.g., the list

  ["x", "y", "z", "y", "y", "z", "z", "x", "a", "y", "y", "z", "z"]

contains 5 "y"s and 5 "z"s, so we should return "y"

## Test2D

- Given a list of strings, sort the list from shortest string to longest string, and in alphabetic order.

- e.g., given the list

  ["abstract", "boolean", "char", "code", "for"]

  should be transformed into the list

  ["for", "char", "code", "boolean", "abstract"]

# Test2G

- Given two strings, determine if they are anagrams. Two strings are anagrams if you can rearrange the letters of one string to form the other string (with no letters left over)

- e.g.,

"pools" and "loops" are anagrams

but

"return" and "nature" are not anagrams

## Test2H

- Given a list of strings, return the set of characters that are not contained in the strings of the list. The returned set is in sorted (alphabetic) order.

- e.g., given the list

["z", "xwv", "tsr", "pon", "lkj", "hgf", "dcba"]

return the set

['e', 'i', 'm', 'q', 'u', 'y']

## Solution strategies

- all questions are solvable using Map
- C, G, and H have reasonable solutions that do not use Map
- D has an elegant solution that does not use Map but requires you to create another class that implements Comparator

## Test2C

```java
public static String mostFrequent(List<String> t) {
  Map<String, Integer> stringCount = new TreeMap<String, Integer>();
  for (String s : t) {
    Integer count = stringCount.get(s);
    if (count == null) {
      count = 0;
    }
    stringCount.put(s, count + 1);
  }
  int max = 0;
  String result = "";
  for (Map.Entry<String, Integer> e : stringCount.entrySet()) {
    int count = e.getValue();
    if (count > max) {
      max = count;
      result = e.getKey();
    }
  }
  return result;
}
```

# Test2D

```java
public static void sortByLength(List<String> t) {
  Map<Integer, List<String>> lengths = new TreeMap<Integer, List<String>>();
  for (String s : t) {
    int sLen = s.length();
    List<String> u = lengths.get(sLen);
    if (u == null) {
      u = new ArrayList<String>();
      lengths.put(sLen, u);
    }
    u.add(s);
  }
  t.clear();
  for (List<String> u : lengths.values()) {
    Collections.sort(u);
    t.addAll(u);
  }
}
```

## Test2G

```
public static boolean areAnagrams(String s, String t) {
  if (s.length() != t.length()) {
    return false;
  }
  List<Character> sList = new ArrayList<Character>();
  List<Character> tList = new ArrayList<Character>();
  for (int i = 0; i < s.length(); i++) {
    sList.add(s.charAt(i));
    tList.add(t.charAt(i));
  }
  Collections.sort(sList);
  Collections.sort(tList);
  return sList.equals(tList);
}
```

## Test2H

```
public static Set<Character> missingLetters(List<String> t) {
  Set<Character> letters = new HashSet<Character>();
  for (char c = 'a'; c <= 'z'; c++) {
    letters.add(c);
  }
  for (int i = 0; i < t.size(); i++) {
    String s = t.get(i);
    for (int j = 0; j < s.length(); j++) {
      char cj = s.charAt(j);
      letters.remove(cj);
    }
  }
  Set<Character> missing = new TreeSet<Character>(letters);
  return missing;
}
```