EECS1022 Summer 2019 Test 3 Version A

Q1: Examine the fragment shown below in which `m` is an initialized variable of type `int` and `list` refers to an empty list of type `ArrayList<String>`. Given that the fragment's output is `abcd`, what is the initial value of the variable `m`?

```
list.add("ab"); list.add("abc"); list.add("ab");
list.add(m - list.size(),"abcd");
System.out.println(list(get(2));
```

Write the answer without adding anything to it (such as extra quotes, leading or trailing text, = signs, ...). Write XXX in case of errors.

Q2: Examine the fragment shown below in which `m` is an initialized variable of type `int` and `set` refers to an empty set of type `TreeSet<Integer>`. Given that the fragment's output is 10, what is the initial value of the variable `m`?

```
set.add(45); set.add(21); set.add(45);
boolean b = set.add(57);
if (b)
{
     System.out.println(m - set.size());
}
else
{
     System.out.println(m + set.size());
}
```

Write the answer without adding anything to it (such as extra quotes, leading or trailing text, = signs, ...). Write XXX in case of errors.

Q3: Examine the fragment shown below in which `m` is an initialized variable of type `int` and `map` refers to an empty map of type `TreeMap<Integer,Integer>`. Given that the fragment's output is 12, what is the initial value of the variable `m`?

```
map.put(2, 3); map.put(7, 5); map.put(2, m);
if (map.containsKey(6))
{
     System.out.println(map.get(2) + map.get(7));
}
else
{
     System.out.println(map.get(2) - map.get(7));
}
```

Write the answer without adding anything to it (such as extra quotes, leading or trailing text, = signs, ...). Write XXX in case of errors.

Q4: Which of the following statements is *true*?

- Both `List<String>` and `Set<String>` represent ordered collections.
- `List<String>` represents ordered collections but `Set<String>` represents unordered collections.
- `List<String>` represents unordered collections but `Set<String>` represents ordered collections.
- Both `List<String>` and `Set<String>` represent unordered collections.
- none of the above

Q5: Which of the following statements is *false*?

- Both `HashSet<String>` and `TreeSet<String>` implement all the methods in the `Set<String>` interface.
- `TreeSet<String>` uses a binary tree representation of sets of strings.
- `HashSet<String>` uses a hash table representation of sets of strings.
- For any method in `Set<String>`, the versions of the method defined by `HashSet<String>` and that defined by `TreeSet<String>` have the same computational compexity.
- none of the above

Q6: Given two non-null, non-empty sets `a` and `b`, the method below returns the set of:

```
public static int work(Set<String> a, Set<String> b)
{
     Set<String> s = new HashSet<String>();
     for (String e : a)
```

```
    {
        if (b.contains(e))
        {
            s.add(e);
        }
    }
    return s;
}
```

- elements in a but not in b
- elements in b but not in a
- elements in the intersection of a and b
- elements in the union of a and b
- none of the above

Q7: The method below returns:

```
public static int m(Map<Integer,Integer> map)
{
    int edge = Integer.MAX_VALUE;
    for (int key : map.keySet())
    {
        if (map.get(key) < edge)
        {
            edge = map.get(key);
        }
    }
    return edge;
}
```

- the largest value in the map
- the largest key in the map
- the smallest value in the map
- the smallest key in the map
- none of the above

Q8: The worst-case complexity of the method below is (`N` is the size of the collection):

```
public static int meth(List<String> list1, List<String> list2)
{
    int result = 0;
    for (String a : list1)
    {
```

```
            if (list2.contains(a))
            {
                  result++;
            }
      }
      return result;
}
```

- O(1)
- O(N)
- O(NlgN)
- $O(N^2)$
- none of the above

Q9: Which of the following is the set of all strings that match the regular expression `^ab?([cd]|e)f$` ?

- { "acf", "adf", "abcf", "abdf", "ef" }
- { "acef", "adef", "abcef", "abdef" }
- { "acf", "adf", "abcf", "abdf", "aef", "abef" }
- { "cf", "df", "abcf", "abdf", "ef" }
- none of the above

Q10: Which of the following string does *not* contain a match for the regular expression `^xy*` ?

- "xy"
- "xxy"
- "xyy"
- "xyyz"
- none of the above

Q11: Implement the method below which takes a list of integers `list`, and returns the sum of all the elements in the list.

For example, if `list` has the value `[7, 13, 9, 13, -8]`, then the method returns the value `34`.

Note: After you develop and test, copy and paste the indicated method only, starting with the shown header and ending with the closing brace. Your code must compile when placed in a class that has only two imports at its top:

```
import java.util.regex.*;
import java.util.*;
```

```
public static int sum(List<Integer> list)
{

}
```

Q12: Implement the method below which takes a string `s` and a list of strings `list`, and returns the list of all the elements in `list` that come before `s` in the lexicographic order (if there are no elements in `list` that come before `s` in the lexicographic order, it returns the empty list). The argument `list` should not be changed by the method.

For example, if `s` has the value `"john"` and `list` has the value `[john, ann, paul, ted, helen]`, then the method returns the value `[ann, helen]`.

Note: After you develop and test, copy and paste the indicated method only, starting with the shown header and ending with the closing brace. Your code must compile when placed in a class that has only two imports at its top:

```
import java.util.regex.*;
import java.util.*;
```

```
public static List<String> select(String s, List<String> list)
{

}
```

Q13: Implement the method below which takes a string `s`, and returns a list of all the serial numbers substrings in `s`; if there is no serial number in `s`, it returns an empty list.

A serial number is a string that starts with "SN" followed by one or more digits followed by two upper case letters.

For example, when `s` has the value `"I bought SN123AB and SN4CE. I already had SNAB, SN57ER, and SN4C. I returned S45AB."`, the returned value should be `[SN123AB, SN4CE, SN57ER]`.

Note: After you develop and test, copy and paste the indicated method only, starting with the shown header and ending with the closing brace. Your code must compile when placed in a class that has only two imports at its top:

```
import java.util.regex.*;
import java.util.*;
```

```
public static List<String>   findAllSerialNos(String s)
{

}
```

Q14: Implement the method below which takes an integer `k` and a map from integers to strings `map`, and returns a set containing all the values in `map` whose length is equal to `k`.

For example, if `k` has the value `3` and `map` has the value `{12=john,15=ann, 17=paul, 18=ted, 21=helen}`, then the method returns the value `[ann, ted]`.

Note: After you develop and test, copy and paste the indicated method only, starting with the shown header and ending with the closing brace. Your code must compile when placed in a class that has only two imports at its top:

```
import java.util.regex.*;
import java.util.*;
```

```
public static Set<String>  filter(int k, Map<Integer,String> map)
{

}
```

Q15: Implement the method below which takes a map from strings to sets of strings `m1` and a map from strings to sets of strings `m2`, and returns a new map that is the intersection of `m1` and `m2`, that is, whose keys are those that are keys of both `m1` and `m2`, and such that the value of a key is the intersection of its value under `m1` and its value under `m2`.

For example, if `m1` is the map `{AA=[aa, bb, cc, dd], BB=[ff, gg], DD=[ii,jj], EE=[mm]}` and `m2` is the map `{AA=[bb, dd, ee], CC=[hh], DD=[jj, kk], EE=[nn,oo]}`, the returned value should be `{AA=[bb, dd], DD=[jj], EE=[]}`.

The method should not make any modification to its arguments `m1` and `m2`.

Note: After you develop and test, copy and paste the indicated method only, starting with the shown header and ending with the closing brace. Your code must compile when placed in a class that has only two imports at its top:

```
import java.util.regex.*;
import java.util.*;
```

```
public static Map<String, Set<String>> mapIntersect(Map<String, Set<String>>m1, Map<String,Set<String>> m2)
{

}
```

Logout