HW-4

1.

Write each of the following *pure* functions (a function is pure if it does not update any of the arguments or global variables).

- `subset(s1,s2)` is true iff `s1` is a subset of `s2`.

```
public static <E> boolean subset(HashSet<E> s1, HashSet<E> s2)
```

- `union(s1,s2)` returns the union of `s1` and `s2`.

```
public static <E> HashSet<E> union(HashSet<E> s1, HashSet<E> s2)
```

- `intersection(s1,s2)` returns the intersection of `s1` and `s2`.

```
public static <E> HashSet<E> intersection(HashSet<E> s1, HashSet<E> s2)
```

- `difference(s1,s2)` returns the difference of `s1` and `s2` (i.e., `s1 − s2`)

```
public static <E> HashSet<E> difference(HashSet<E> s1, HashSet<E> s2)
```

- `cartesianProduct(s1,s2)` returns the Cartesian product of `s1` and `s2`.

```
public static <U,V> HashSet<Pair<U,V>>
                    cartesianProduct(HashSet<U> s1, HashSet<V> s2)
```

  where `Pair` is defined as follows:

```
class Pair<U,V>{
    public U first;
    public V second;
    public Pair(U first, V second){
        this.first = first;
        this.second = second;
    }
}
```

2.

The following implementation of the function `removeDuplicates(list)` does not scale well due to its quadratic time complexity.

```
public static <E> ArrayList<E> removeDuplicates(ArrayList<E> list) {
    ArrayList<E> newList = new ArrayList<>();
    for(int i = 0; i < list.size(); i++) {
        if(!newList.contains(list.get(i))) {
            newList.add(list.get(i));
        }
    }
    return newList;
}
```

Improve the implementation by using a set to make it more scalable.

3. (project)

The program given below count keywords in a Java source file. Revise the program such that keywords in comments are disregarded.

```
import java.util.*;
import java.io.*;

public class CountKeywords {
  public static void main(String[] args) throws Exception {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a Java source file: ");
    String filename = input.nextLine();

    File file = new File(filename);
    if (file.exists()) {
      System.out.println("The number of keywords in " + filename
        + " is " + countKeywords(file));
    }
    else {
      System.out.println("File " + filename + " does not exist");
    }
  }

  public static int countKeywords(File file) throws Exception {
    // Array of all Java keywords + true, false and null
    String[] keywordString = {"abstract", "assert", "boolean",
        "break", "byte", "case", "catch", "char", "class", "const",
        "continue", "default", "do", "double", "else", "enum",
        "extends", "for", "final", "finally", "float", "goto",
        "if", "implements", "import", "instanceof", "int",
        "interface", "long", "native", "new", "package", "private",
        "protected", "public", "return", "short", "static",
        "strictfp", "super", "switch", "synchronized", "this",
        "throw", "throws", "transient", "try", "void", "volatile",
        "while", "true", "false", "null"};
```

```
    Set<String> keywordSet =
      new HashSet<>(Arrays.asList(keywordString));
    int count = 0;

    Scanner input = new Scanner(file);

    while (input.hasNext()) {
      String word = input.next();
      if (keywordSet.contains(word))
        count++;
    }

    return count;
  }
}
```

4. (project)

The program shown below counts the occurrences of words in a text, and displays the words and their occurrences in alphabetical order. Suppose you have a dictionary that maps words of different forms to their root forms. For example, the dictionary maps verbs *see*, *sees*, *seeing*, *saw*, and *seen* to their root form *see*. Write a program that counts the occurrences of words in a text, treating all forms of a root word as the root word.

```
import java.util.*;

public class CountOccurrenceOfWords {
  public static void main(String[] args) {
    // Set text in a string
    String text = "Good morning. Have a good class. " +
      "Have a good visit. Have fun!";

    // Create a TreeMap to hold words as key and count as value
    Map<String, Integer> map = new TreeMap<>();

    String[] words = text.split("[\\s+\\p{P}]");
    for (int i = 0; i < words.length; i++) {
      String key = words[i].toLowerCase();

      if (key.length() > 0) {
        if (!map.containsKey(key)) {
          map.put(key, 1);
        }
        else {
          int value = map.get(key);
          value++;
          map.put(key, value);
```

```
        }
      }
    }

    // Display key and value for each entry
    map.forEach((k, v) -> System.out.println(k + "\t" + v));
  }
}
```