```java
package androidPermissions;

public class Permissions {
    int count;
    String type;

    public Permissions(int c, String t) {
        this.count = c;
        this.type = t;
    }

    public void increaseCount() {
        count++;
    }

    public int getCount() {
        return count;
    }

    public String getType() {
        return type;
    }
}

```

```java
package androidPermissions;

import java.io.*;
import java.util.*;
import java.util.Map.Entry;
import org.apache.commons.collections4.ListUtils;

public class AnalyzeAndroidPermissions {
    static HashSet<String> normal = new HashSet<>();
    static HashSet<String> dangerous = new HashSet<>();
    static HashSet<String> signature = new HashSet<>();
    static HashSet<String> removed = new HashSet<>();
    static HashSet<String> other = new HashSet<>();
    static File[] spyFiles = new File("src/res/spyware").listFiles();
    static File[] popFiles = new File("src/res/popular").listFiles();
    static File permFile = new File("src/res/permissions/allPermissions.txt");
    static File spyData = new File("src/res/permissions/spywarePermissions.txt");
    static File popData = new File("src/res/permissions/popularPermissions.txt");
    static File permByAppFile = new File("src/res/permissions/permissionsByApp.txt");
    static Map<String, Permissions> spyMap = new HashMap<>();
    static Map<String, Permissions> popMap = new HashMap<>();
    static File spyCount = new File("src/res/permissions/spywarePermissionsCount.txt");
    static File popCount = new File("src/res/permissions/popPermissionsCount.txt");
    static File sharedPerms = new File("src/res/permissions/sharedPermissions.txt");
    static File uniquePerms = new File("src/res/permissions/uniquePermissions.txt");

    public static void main(String[] args) {
        try {
            getPermissions(spyFiles, spyData, permByAppFile);
            getPermissions(popFiles, popData, permByAppFile);
            getCategories();
            List<Entry<String, Permissions>> sortedSpy = countPermissions(spyData, spyMap, spyCount);
            List<Entry<String, Permissions>> sortedPop = countPermissions(popData, popMap, popCount);
            comparePermissions(sortedSpy, sortedPop);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Reads all permissions from set of apps (spyware or popular) and writes to file
    static void getPermissions(File[] files, File dataTxt, File labeledTxt) throws IOException {
        String line;
        String prefix = "<uses-permission android:name=";
        BufferedWriter bw1 = new BufferedWriter(new FileWriter(dataTxt));
        BufferedWriter bw2 = new BufferedWriter(new FileWriter(labeledTxt, true));

        for (File file : files) {
            BufferedReader br = new BufferedReader(new FileReader(file));
            // Used to filter out repeated permissions within the same XML file
            HashSet<String> uniquePerms = new HashSet<>();
            bw2.write(file.toString() + "\n");
            while ((line = br.readLine()) != null) {
                if (line.contains(prefix)) {
                    String[] result = line.split("\"");
                    if (!uniquePerms.contains(result[1])) {
                        bw1.write(result[1] + "\n");
                        bw2.write(result[1] + "\n");
                        uniquePerms.add(result[1]);
                    }
                }
            }
            bw2.write("\n");
            br.close();
        }
        bw1.close();
        bw2.close();
    }
    // Reads categorized permissions (normal, dangerous, signature, or removed) and adds to set for use in
program
    static void getCategories() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(permFile));
        String line;

        if ((br.readLine()).equals("Dangerous")) {
            while (!(line = br.readLine()).equals("")) {
                dangerous.add(line);
            }
        }
        if ((br.readLine()).equals("Normal")) {
```

```
 78             while (!(line = br.readLine()).equals("")) {
 79                 normal.add(line);
 80             }
 81         }
 82         if ((br.readLine()).equals("Signature")) {
 83             while (!(line = br.readLine()).equals("")) {
 84                 signature.add(line);
 85             }
 86         }
 87         if ((br.readLine()).equals("Removed")) {
 88             while (!(line = br.readLine()).equals("")) {
 89                 removed.add(line);
 90             }
 91         }
 92         if ((br.readLine()).equals("Other")) {
 93             while ((line = br.readLine()) != null) {
 94                 other.add(line);
 95             }
 96         }
 97         br.close();
 98     }
 99     // Counts repeated permissions and outputs to file sorted by type and descending count
100     static List<Entry<String, Permissions>> countPermissions(File txt, Map<String, Permissions> map, File
    countTxt)
101             throws IOException {
102         String line;
103         boolean isNormal = false;
104         boolean isDangerous = false;
105         boolean isSignature = false;
106         boolean isRemoved = false;
107         boolean isOther = false;
108         BufferedReader br = new BufferedReader(new FileReader(txt));
109         BufferedWriter bw = new BufferedWriter(new FileWriter(countTxt));
110 
111         while ((line = br.readLine()) != null) {
112             if (map.containsKey(line)) {
113                 map.get(line).increaseCount();
114             } else {
115                 if (normal.contains(line)) {
116                     map.put(line, new Permissions(1, "normal"));
117                     isNormal = true;
118                 }
119                 if (!isNormal) {
120                     if (dangerous.contains(line)) {
121                         map.put(line, new Permissions(1, "dangerous"));
122                         isDangerous = true;
123                     }
124                 }
125                 if (!isNormal && !isDangerous) {
126                     if (signature.contains(line)) {
127                         map.put(line, new Permissions(1, "signature"));
128                         isSignature = true;
129                     }
130                 }
131                 if (!isNormal && !isDangerous && !isSignature) {
132                     if (removed.contains(line)) {
133                         map.put(line, new Permissions(1, "removed"));
134                         isRemoved = true;
135                     }
136                 }
137                 if (!isNormal && !isDangerous && !isSignature && !isRemoved) {
138                     if (other.contains(line)) {
139                         map.put(line, new Permissions(1, "other"));
140                         isOther = true;
141                     }
142                 }
143                 if (!isNormal && !isDangerous && !isSignature && !isRemoved && !isOther) {
144                     map.put(line, new Permissions(1, "unknown"));
145                 }
146                 isNormal = false;
147                 isDangerous = false;
148                 isSignature = false;
149                 isRemoved = false;
150                 isOther = false;
151             }
152         }
153         br.close();
154 
```

```
155         List<Entry<String, Permissions>> sortedEntries = new ArrayList<>(map.entrySet());
156         Collections.sort(sortedEntries, Entry.comparingByValue(Comparator.comparing(Permissions::getType)
157                 .thenComparing(Permissions::getCount, Comparator.reverseOrder())));
158 
159         for (Entry<String, Permissions> entry : sortedEntries) {
160             int count = entry.getValue().getCount();
161             String type = entry.getValue().getType();
162             String name = entry.getKey();
163             double percentIncl = ((double)count / 20) * 100;
164 
165             bw.write(count + " " + type + " " + name + " " + String.format("%,.0f", percentIncl) + "%\n");
166         }
167         bw.close();
168 
169         return sortedEntries;
170     }
171     // Returns shared and unique permissions
172     static void comparePermissions(List<Entry<String, Permissions>> sp, List<Entry<String, Permissions>> pp)
173             throws IOException {
174         BufferedWriter bw1 = new BufferedWriter(new FileWriter(sharedPerms));
175         BufferedWriter bw2 = new BufferedWriter(new FileWriter(uniquePerms));
176         List<String> spyList = new ArrayList<>();
177         List<String> popList = new ArrayList<>();
178         List<String> sharedList;
179         List<String> uniqueSpyList;
180         List<String> uniquePopList;
181         String type;
182         double percentIncl;
183         double spyPer;
184         double popPer;
185 
186         for (Entry<String, Permissions> e : sp) {
187             spyList.add(e.getKey());
188         }
189 
190         for (Entry<String, Permissions> e : pp) {
191             popList.add(e.getKey());
192         }
193 
194         sharedList = ListUtils.intersection(spyList, popList);
195         uniqueSpyList = ListUtils.subtract(spyList, popList);
196         uniquePopList = ListUtils.subtract(popList, spyList);
197         Collections.sort(sharedList);
198         Collections.sort(uniqueSpyList);
199         Collections.sort(uniquePopList);
200 
201         for (String e : sharedList) {
202             type = spyMap.get(e).getType();
203             spyPer = ((double)(spyMap.get(e).getCount()) / 20) * 100;
204             popPer = ((double)(popMap.get(e).getCount()) / 20) * 100;
205             percentIncl = ((double)(spyMap.get(e).getCount() + popMap.get(e).getCount()) / 40) * 100;
206             bw1.write(type + " " + e + " " + String.format("%,.0f", spyPer) + "% "
207                     + String.format("%,.0f", popPer) + "% " + String.format("%,.0f", percentIncl) +"%\n");
208         }
209 
210         bw2.write("Spyware\n");
211         for (String e : uniqueSpyList) {
212             type = spyMap.get(e).getType();
213             percentIncl = ((double)spyMap.get(e).getCount() / 20) * 100;
214             bw2.write(type + " " + e + " " + String.format("%,.0f", percentIncl) +"%\n");
215         }
216         bw2.write("\nPopular\n");
217         for (String e : uniquePopList) {
218             type = popMap.get(e).getType();
219             percentIncl = ((double)popMap.get(e).getCount() / 20) * 100;
220             bw2.write(type + " " + e + " " + String.format("%,.0f", percentIncl) +"%\n");
221         }
222 
223         bw1.close();
224         bw2.close();
225     }
226 }
227 
```