Write a method named **`arrangements`** that accepts a `List` of names as a parameter and prints out all of the possible arrangements of the people in a line.

For example, suppose a `List` called *list* stored the following names:

```
["alice", "bob", "carl"],
```

Then a call of `arrangements(list)` should produce the following output:

```
[alice, bob, carl]
[alice, carl, bob]
[bob, alice, carl]
[bob, carl, alice]
[carl, alice, bob]
[carl, bob, alice]
```

The order in which you show each of the arrangements does not matter. The key thing is that your method should produce the correct overall set of arrangements as its output. You may assume that the list passed to your method is not null and that the list contains no duplicates.

Hint: This is very similar to permute from lecture!

One possible solution:

```
public static void arrangements(List<String> people) {
      arrangements(people, new ArrayList<String>());
}

public static void arrangements(List<String> people, List<String> chosen) {
      if (people.size() == 0) {
            System.out.println(chosen);
      } else {
            for (int i = 0; i < people.size(); i++) {
                  String person = people.remove(i);
                  chosen.add(person);
                  arrangements(people, chosen);
                  chosen.remove(chosen.size() - 1);
                  people.add(i, person);
            }
      }
}
```

Write a method named **`arrangements2`** that accepts a `List` of names, and two Strings `name1` and `name2` as a parameter and prints out all of the possible arrangements of the people in a line where `name1` and `name2` are not next to each other.

For example, suppose a `List` called *list* stored the following names:

```
["alice", "bob", "carl"],
```

Then a call of `arrangements(list, "alice", "bob)` should produce the following output:

```
[alice, carl, bob]
[bob, carl, alice]
```

The order in which you show each of the arrangements does not matter. The key thing is that your method should produce the correct overall set of arrangements as its output. You may assume that the list passed to your method is not null and that the list contains no duplicates. If there are no ways to arrange the names in the list such that `name1` and `name2` are not adjacent, then your method should produce no output.

Hint: Start with the previous problem, arrangements, and modify it to meet this additional constraint

Two possible solutions:

```
public static void arrangements2(List<String> people, String p1, String p2) {
   arrangements2(people, new ArrayList<String>(), p1, p2);
}

public static void arrangements2(List<String> people, List<String> chosen,
                        String p1, String p2) {
   if (people.size() == 0) {
      // check that p1 and p2 aren't next to each other before printing
      int position = Math.abs(chosen.indexOf(p1) - chosen.indexOf(p2));
      if (position != 1) {
         System.out.println(chosen);
      }
   } else {
      // try each of the people left as the next person
      for (int i = 0; i < people.size(); i++) {
         String person = people.remove(i);
         chosen.add(person);
         arrangements2(people, chosen, p1, p2);
         chosen.remove(chosen.size() - 1);
         people.add(i, person);
      }
   }
}
```

```
public static void arrangements2(List<String> people, String p1, String p2) {
   arrangements2(people, new ArrayList<String>(), p1, p2);
}

public static void arrangements2(List<String> people, List<String> chosen,
                        String p1, String p2) {
   if (people.size() == 0) {
      System.out.println(chosen);
   } else {
      // try each of the people left as the next person
      for (int i = 0; i < people.size(); i++) {
         String person = people.remove(i);
         // Only choose this person if this wouldn't put p1 and p2
         // next to each other. e.g. the previous person isn't p1 and
         // this person p2 and vice versa
         int lastIndex = chosen.size() - 1;
         if (chosen.isEmpty() ||
              !(chosen.get(lastIndex).equals(p1) && person.equals(p2)) &&
              !(chosen.get(lastIndex).equals(p2) && person.equals(p1))) {
            chosen.add(person);
            arrangements2(people, chosen, p1, p2);
            chosen.remove(chosen.size() - 1);
         }
         people.add(i, person);
      }
   }
}
```