Lab 10

1. A *group* is a collection of elements having one special element. An example
   [special : 1, elements : [1, 2, 3, 4]]
   [special : 8, elements : [4, 8, 5, 6]]
   [special : “Java”, elements : [“C++”,”C#”, ”Java”,”Kotlin”]]

   Here is a representation of a group as a Java class:

```
public class Group<T> {
    private T specialElement;
    private List<T> elements = new ArrayList<>();
    public Group(T special, List<T> elements) {
        this.specialElement = special;
        this.elements = elements;
    }
}
```

   The following static method attempts to make a copy of a given instance of a Group, reproducing the state of the group in the copy.

```
public static Group<?> copy(Group<?> group) {
    List<?> elements = group.getElements();
    Group<?> grp = new Group<?>(group.getSpecialElement(), elements);
    return grp;
}
```

   The code does not compile. Fix the code by capturing the wildcard with a helper method. Startup code is provided in the directory for this lab problem. Use the main method provided there to test your implementation. Note that the Group class has a toString method that will help in your test.

2. Create a generic programming solution to the problem of finding the second smallest element in a list. In other words, devise a public static method secondSmallest so that it can handle the biggest possible range of types.

3. Generalize the `contains` method for a `List` in the following way. First consider a simple implementation for a `List` of `Strings`:

```
public static boolean contains1(List<String> list, String s) {
    for(String x: list) {
        if(x == null && s == null) return true;
        if(s == null || x == null) continue;
        if(x.equals(s)) return true;
    }
    return false;
}
```

This `contains` method is tested in the following test method:

```
public static void test1() {
    List<String> list = Arrays.asList("Bob", "Joe", "Tom");
    boolean result = Main.contains1(list, "Tom");
    System.out.println(result);
}
```

We would like to generalize to a type variable T. Write the code for the most general possible `contains` method so that the type T can represent Employees, Account and other types.