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. Write a generic method to find tht sum of List elements. Your sum() method need to return the double value.

4. 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);
}
```

In more general lists, the objects in the List may not have overridden the equals method. This could be handled by introducing a BiPredicate, as in the following:

```
public static boolean contains2(List<Employee> list, Employee e,
        BiPredicate<Employee,Employee> pred2) {
    for(Employee emp: list) {
        if(emp == null && e == null) return true;
        if(emp == null || e == null) continue;
        if(pred2.test(emp, e)) return true;
    }
    return false;
}
```

The BiPredicate can be used to represent an equals method for Employees. The following test uses it in this way, declaring two Employees to be equal if their id's are equal:

```
public static void test2() {
    List<Employee> list = new ArrayList<>();
    list.add(new Employee(1003, "Tom", 60000));
    list.add(new Employee(1002, "Harry", 70000));
    list.add(new Employee(1001, "Joe", 50000));
    Employee e = new Employee(1001, "Joe", 50000);
    boolean foundIt = Main.contains2(list, e, (e1,e2) -> e1.getId()==e2.getId());
    System.out.println(foundIt);
}
```

Now we want to generalize from Employee to a type variable T. Write the code for the most general possible contains method. Test your method using each of the following test() methods. You should be able to define a single contains method that will give correct results for each of these tests. Startup code is in the code folder for this lab.

```
public static void test2() {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee(1003, "Tom", 60000));
        list.add(new Employee(1002, "Harry", 70000));
        list.add(new Employee(1001, "Joe", 50000));
        Employee e = new Employee(1001, "Joe", 50000);
        boolean foundIt = Main.contains2(list, e,
                (e1,e2) -> e1.getId()==e2.getId());
        System.out.println(foundIt);
}


public static void test3() {
        List<Manager> list = new ArrayList<>();
        list.add(new Manager(1003, "Tom", 60000, 700));
        list.add(new Manager(1002, "Harry", 70000, 400));
        list.add(new Manager(1001, "Joe", 50000, 500));
        Manager m = new Manager(1001, "Joe", 50000, 500);
        boolean foundIt = Main.contains3(list, m,
    (Employee e1, Employee e2) -> e1.getId()==e2.getId());
        System.out.println(foundIt);
}
```

```
public static void test4() {
        List<Manager> list = new ArrayList<>();
        list.add(new Manager(1003, "Tom", 60000, 700));
        list.add(new Manager(1002, "Harry", 70000, 400));
        list.add(new Manager(1001, "Joe", 50000, 500));
        Manager m = new Manager(1001, "Joe", 50000, 500);
        boolean foundIt = Main.contains3(list, m,
     (Employee e, Person p) -> e.getName().equals(p.getName()));
        System.out.println(foundIt);
}

public static void test5() {
        List<CheckingAccount> list = new ArrayList<>();
        list.add(new CheckingAccount(1001, 25.00));
        list.add(new CheckingAccount(1002, 35.00));
        list.add(new CheckingAccount(1003, 125.00));
        Account a = new CheckingAccount(1002, 35.00);
        boolean foundIt = Main.contains4(list, a,
    (Account a1, Account a2) -> a1.getAcctId()==a2.getAcctId());
        System.out.println(foundIt);
}
```