1. 1. Java’s ArrayList implements following 6 interfaces
      1. [Serializable](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Serializable.html),
      2. [Cloneable](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Cloneable.html)
      3. [Iterable](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html)<E>
      4. [Collection](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Collection.html)<E>
      5. [List](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html)<E>
      6. [RandomAccess](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/RandomAccess.html)

And it extends one class, [AbstractList](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/AbstractList.html)<E>.

* 1. Employee class’s equals(Employee e) method is not overridden. It’s just that class’s own method. listsAreEqual(List<Employee> l1, List<Employee> l2) method in EmployeeInfo class used contains method. It’s checked with the Object class’s original equals method. So, we got incorrect results. So, to make it correct, we have to override equals properly in Employee class. Solution is in the code package.
  2. Employee class needs to override hashCode() method. Because it’s not overridden yet, the result is being incorrect at removeDuplicates method while using HashMap. Solution is in the code package.
  3. In the EmployeeInfo class’s removeDuplicates method, object value, setVisited, was changed. Although now it’s getting correct results, it will lead to incorrect results in some cases. To use an object as a key in a hashtable, keys should be immutable. Solution is in the code package.
  4. Java 8 handles the Diamond Problem by the following:
     1. When the type D is a class and A, B, C are interfaces,
        1. Make one of those interfaces as a class and make D extends that class and implements others because any method inherited from a class or a superclass is given higher priority over any default method inherited from an interface.
        2. Derived interfaces or sub-interfaces take higher precedence than the interfaces higher-up in the inheritance hierarchy.
        3. The implementing class, D, has to specifically override and provide a method with the same method definition.

E.g. in D class,

void myMethod(){

B.super.myMethod();

}

* + 1. When the type D is an interface also,
       1. Make D interface extends one of A, B, C interfaces. So, it will inherit an only super interface.
       2. If one of these A, B, C has a default method, any subinterface of both interfaces, D, must provide a default method (i.e. an implementation) of this method, or declare the method (even if unimplemented).