1A. ArrayList interfaces:

- List
- RandomAccess
- Cloneable
- java.io.Serializable
- Iterable
- Collection

ArrayList extends

- AbstractList

1B. In the listsAreEqual function, it had a checking of list.contains for the Employee class however this list.contains is calling the method “equals” from the Object superclass and this was not overridden in the Employee class and it was incorrectly implemented inside the Employee Class. The correct way of doing an equals inside the class should have an Object type parameter instead of the actual Class since this would correctly override the equals method from the Object superclass.

1C. In the removeDuplicates function, the hashmap was defined and the containsKey method was utilised. However there was no hash method override in the Employee class thus resulting in an incorrect output of the equality check. Behind the scenes, the containsKey of hashmap uses element1.hashcode.equals(element2.hashcode) so in order for that to work, we need to define the hashcode method and override it from the Object superclass since if we do not override this method, the default hashCode method calculates different hash codes for different objects with the same value since in hashmap, the hashing would be focused on the key so that in containsKey, it can verify if the key has indeed exist.

1D. In the removeDuplicates function, the property **visited** was mutated resulting to an inconsistent equality check in the listsAreEqual function since the dupsRemoved list has visited set to false by default however in the removeDuplicate list, some values were set to true due to the condition in removeDuplicates thus it will result to an unmatching list. By removing the checking of **visited** in both equals and in hash, it would now return true.

1E. In cases of the diamond problem if D is a class, then class D would need to override which method to use, if either B or C. Also, class D can provide its own implementation for the method also by overriding the method then providing its own implementation. In cases of the diamond problem if D is an interface, then it can either choose which method to use between interface B and C or purely override the default method with its own implementation.