1a. Implements

1. Serializable,
2. Cloneable,
3. Iterable<E>,
4. Collection<E>,
5. List<E>,
6. RandomAccess

Extends: AbstractList<E>

1b. Equals signature was wrong, if we want to override method equals then, we need to use @Override as identifier, and most parameter for equals should be Object.

**public** **boolean** equals(Employee e) {

**return** e.name.equals(name) && e.salary == salary;

}

Correct answer:

1c. Every time we use HashMap for inserting and retrieve would use different hash code value, if we don’t specify hash code ( use default hash code). If we use default hash code, then automatically hash code using memory address, therefore every time we insert and using containsKey(e) they would be different.

Solution:

We need to Override hashCode()

@Override

**public** **int** hashCode() {

**return** Objects.*hash*(**this**.name, **this**.salary);

}

1d. Because on equals method there is no && emp.visited == visited

However on removeDuplicates method the is tracker.get(e).setVisited(**true**);. If we change signature hashCode (in this case add visited) then when we modify we need verify meet with the signature.

Solution:

1. Remove tracker.get(e).setVisited(**true**);.from removeDuplicates method.
2. Or Add emp.visited == visited on equals

1E.

i. When the type D is a class and A, B, C are interfaces

Answer: if both B and C provide default implementations, then they conflict, and D must provide an overriding declaration, possibly using the syntax B.super.method() OR C.super.method() in the body of m to explicitly choose one of the inherited methods

@Override

**public** **void** method() {

// **TODO** Auto-generated method stub

B.**super**.method();

}

Or

@Override

**public** **void** method() {

// **TODO** Auto-generated method stub

C.**super**.method();

}

ii. Because inheritance is possible from multiple interfaces, the same default method can be inherited from different paths. Since each inherited default method provides a different implementation, the compiler needs a way of selecting the declaration to use. This is its method:

* Classes always win. A declaration in the class or a superclass takes priority over any default method declaration.
* Otherwise, the method with the same signature in the *most specific* default-providing interface is selected. For example, based on the rules above the following code will print  **Hello World from B**:

Code attached for both problems.