1) The files Person.java and Person_Bad_Map.java are attached to this assignment. Note the key is age on these records.

a) Explain why the record "Chuck, 18" has gone missing when I do a look up, but is still there when I print out the keySet and values.

When we tried to grab chuck by age, The object of chuck : 30 was changed. overwritten with 18. So the key 30 is lost, the index no longer has the key to the original chuck and since chuck:18 was inserted at the index of chuck:30, searching for chuck 18 does not exist because its in the position of chuck:30 but the key-value is still there. Just not in the right position. When calling chuck:18, chuck:18 is in chuck:30 position so it is not found, but when printing all the key-value pairs, it technically is found because the key-value pair exist, just not at the right position, and pulling that position yields null results, because technically chuck:18 does not have its own position.

b) I was able to change the age of the "Chuck" recored to a new value, and the record for Chuck comes back. But the new value made the record for Ryan disappear. Give me one value (there are several) that causes this problem.

If I change the value of chucks age to 55, then the Ryan object will be gone when I try to find it by its key-value pair

```
p = tm.get(new Person("Chuck", 30));

p.changeAge(55);

Person p1 = tm.get(new Person("Ryan", 55));

System.out.println(p1);
```

```
name: Chuck, age: 55
```

c) Correct this program by creating an immutable "Age" object, and use it to update the table. Keep the Person object mutable, and change the age for a record. Do not use an Integer object for the age, create an Age object. I want to see you follow the steps needed.

Please see attached Person.java

d) Let's create an interesting bug. See if you can make Ryan appear and disappear by just adding records to the table (Extra Credit, I think it will be a little hard).

```
p = tm.get(new Person("Chuck", 30));

p.changeAge(55);

Person p1 = tm.get(new Person("Ryan", 55));

System.out.println(p1);
```

```
      name: Chuck, age: 55
```

```
// Since Chuck is still stored at the index where ryan is at.

// I can add it back to its original index value and pull the record back

p = new Person("Ryan", 55);

tm.put(p, p);

Person p2 = tm.get(new Person("Ryan", 55));

System.out.println(p1);

System.out.println(p2);
```

```
      name: Chuck, age: 55

      name: Ryan, age: 55
```

2) Come up with an example of classification and composition that implements the problem illustrated in the Employee example. The problem at its core is the need when using classification (inheritance) to create a new object that might exist in multiple maps, and that this object can cause multiple copies of the object needing to be managed in the system.

For composition, please see the attached files Career.java , Worker.java, TestWorker.java

For classification please see the attached Car.java , HondaCar.java , Vehicle.java, VehicleTester.java