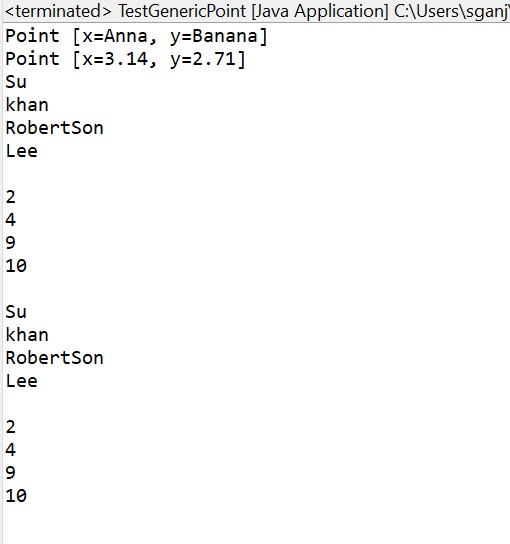
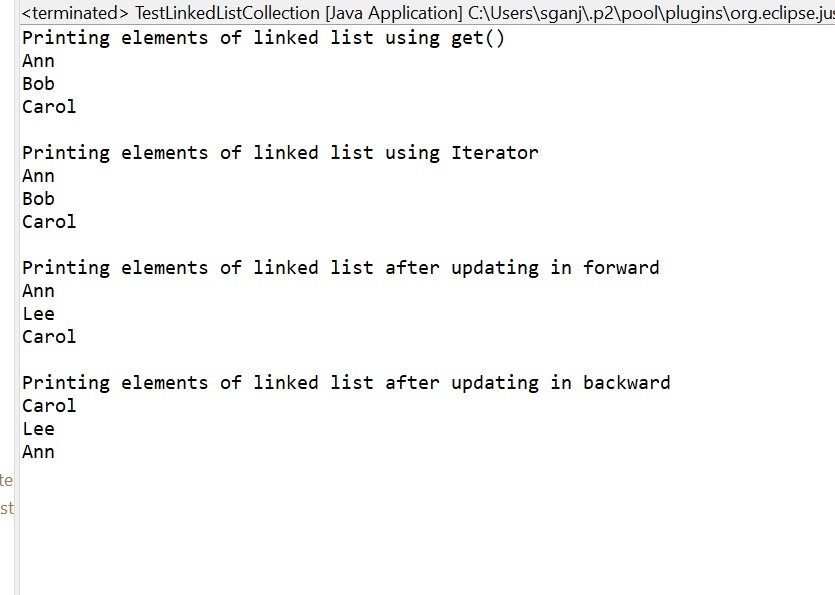
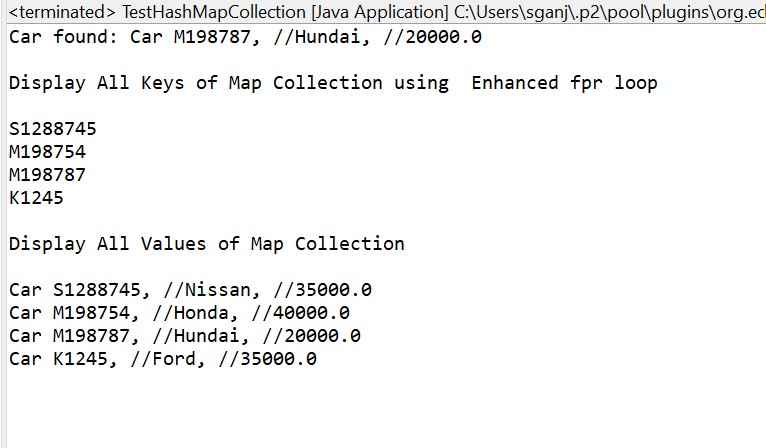
1. Generic methods & Generic classes

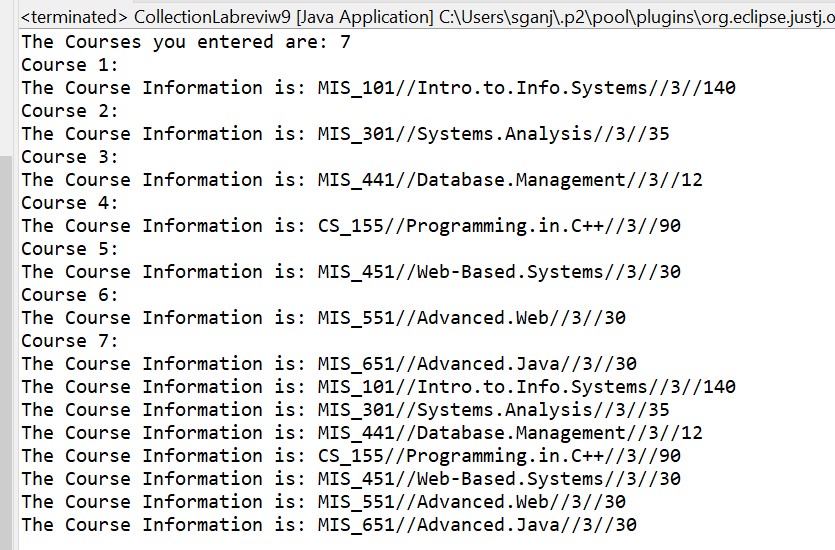


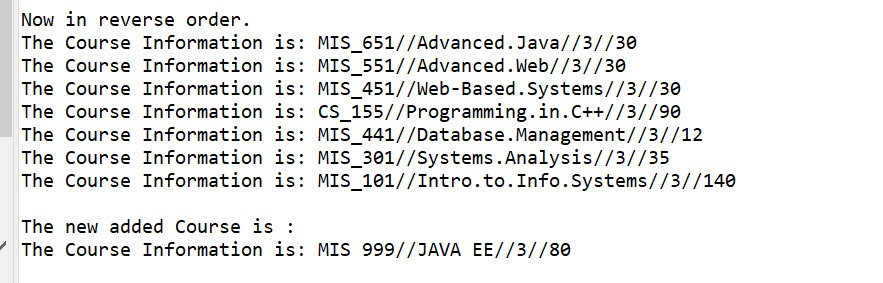
1. Java Collection Framework (LinkedList, ArrayList)



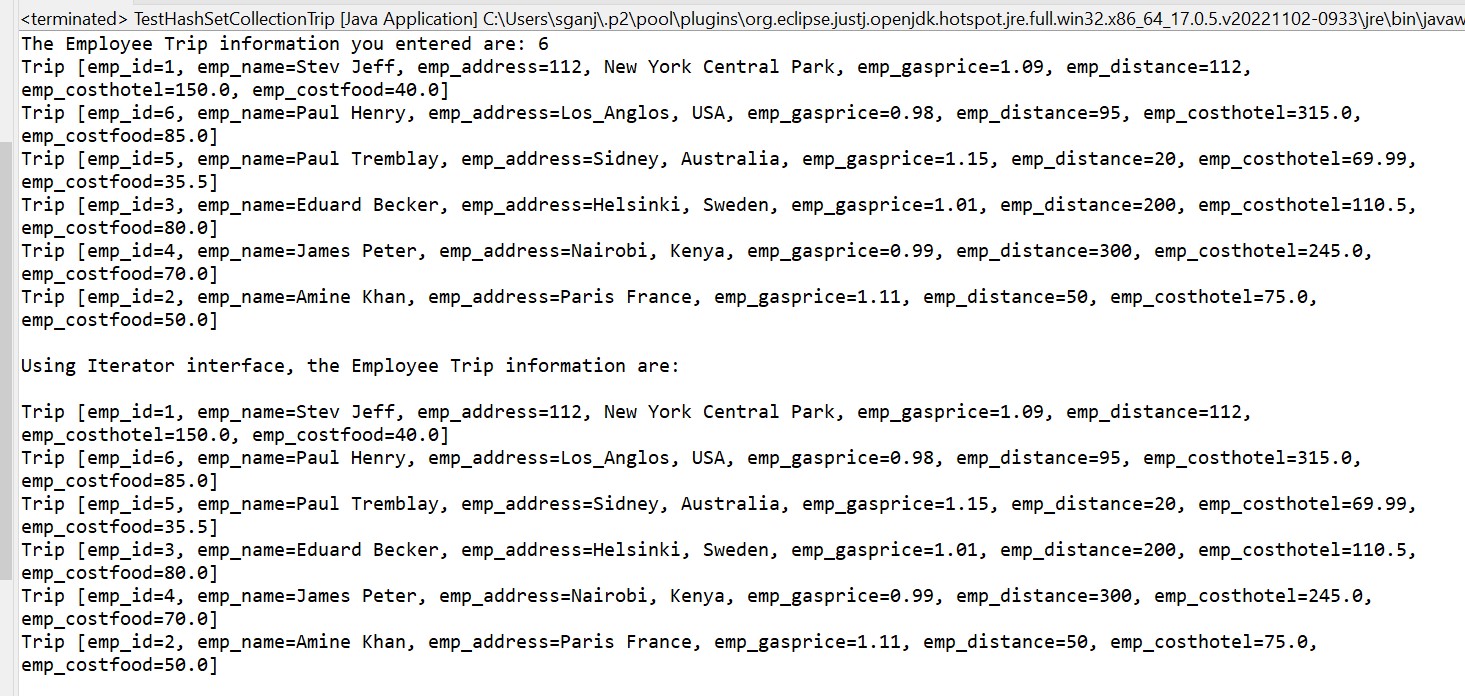


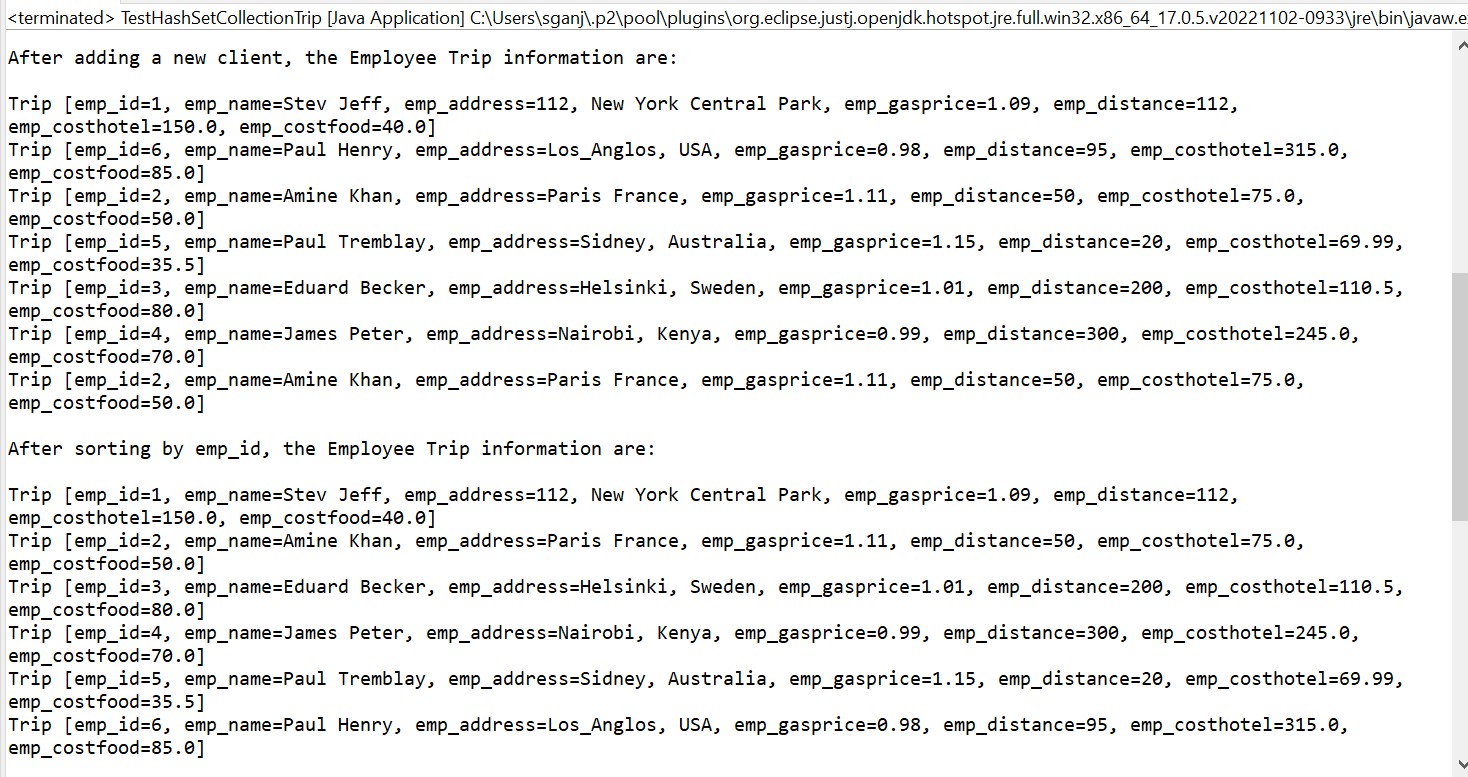
1. Using ArrayList class and its methods add(), size(), get()





1. Using HashSet class and its methods add(), size()







In the given code, the `HashSet` is used to store `Trip` objects. When adding elements to a `HashSet`, it uses the `hashCode()` method of the objects to determine the position to store the element. If two objects have the same hash code, they are added to the same position in the `HashSet`. In this case, the two objects are compared using the `equals()` method to check for equality, and the duplicate is rejected.

In the given code, the `Trip` class seems to have overridden the `hashCode()` and `equals()` methods, which are used by the `HashSet` to compare elements. Therefore, the `HashSet` will only accept `Trip` objects that have unique `hashCode()` values and are not considered equal according to the `equals()` method.

The new `Trip` object added to the `HashSet` has a different `hashCode()` value than any of the objects already in the `HashSet`, and it is not considered equal to any of the existing `Trip` objects, so it is added to the `HashSet` without issue.

It's important to ensure that the `hashCode()` and `equals()` methods are properly implemented for objects that are added to a `HashSet` (or any other set implementation) to avoid unexpected behavior.

1. Using HashMap class and the methods put(), size(), get(), keyset(), values()

