Collections

1) **Insert an element at desired location**
   Take the following inputs from a user:-
   a) Element: e.g. "PQRS"
   b) Desired Location(1-5): e.g. 4

   Create a List<Object> with the following values:-
   {"One",2,"Three",67.986,'Z', "ABC",123,true, "xyz"}

   The 'Desired Location' should only be between 1-5. Any other input should give the following error to the user "Only values between 1-5 are allowed, please try again. and the prompt should be redisplayed to the user.

'Element' can be any type of the following elements:

a) Boolean - If Boolean reverse the value and then add, i.e. if true you add false to the list.
b) Character - if character, you add the next character to the list i.e. if 'C' you add 'D'….if 'c' you add 'd' to the list.
c) String - Change the case of string before adding to the list.
d) Float / Double - Add 2.0 before adding to the list.
e) Integer - Add 1 before adding to the list.

Upon successful addition of an element:-

a) Redisplay all the elements of the list on the console.
b) Return the prompt back to the user so he can enter another element.

When the user types '#' on the console, terminate the program.

Use List for implementation.

2) **Sorting**
   Create a java program which gives the following choices on the command line to the user:-
   1 - Sort by age
   2 - Sort by first name
   3 - Sort by last name

   Once the user has made his/her selection, the system must sort by that parameter a pre-created list of employee objects.
   Employee (Integer id, String firstName, String lastName, Integer age).

   Note:

1) When sorting by first name - If the firstname of two employees are the same, then the system should then sort them by last name.
2) When sorting by last name - If the lastname of two employees are the same, then the system should then sorty them by last name.
3) Display all results on the console.
4) Use List to store all Employees.
5) Write JUnit tests. Coverage should be atleast 70%.

3) Mimic the behavior of an Set (following operations only) using only Arrays.

a) int add(Object o) - Adds only if the object is already not present(for simple object types). On successful addition returns the index no of the location where the new object was added. If duplicate element, -1 should be returned.
b) void remove(Object o) -
c) showAll() - prints all elements to console.
d) Null values should not be allowed.
e) Duplicate elements should not be allowed.
f) Write JUnit tests, Coverage should be atleast 70%.

4) Person class has the following properties:-
Person(Integer id, String name, Float salary)
Create a PersonService class:-

a) List<Person> generateList() - This method should create a list of Person objects. Use any arbitrary data.
b) Map<Float, Person> transform(List<Person> people) - This method takes as input the List<Person> in the previous step and converts it to a Map<Integer, Person>. The key (integer) should be the Person.id. If two or more Person objects in the List<Person> has duplicate ids, then before Inserting in the Map, then append one on the decimal side.
e.g.
PersonA - 1, ankur, 20000 : Map<1, Person(1, ankur, 20000)>
PersonB - 1, ravi, 30000 : Map<1.1, Person<1, ravi, 30000)>
PersonC - 1, prem, 50000: Map<1.11, Person,1,50000>

c) show(List<Person>) - prints the Person list on the console in readable format.
d) show(Map<Float, Person> - prints the Person map on the console in readable format.
e) Write JUnit tests. Coverage should be atleast 70%.