**Learning Platform - Requirement 1**

**Requirement 1:**

Let’s start off by creating two **User** objects and check whether they are equal.

1. Create a **User** class with the following attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| Name | String |
| emailId | String |
| contactNumber | String |

1. Mark all the attributes as private
2. Create / Generate appropriate Getters & Setters
3. Add a default constructor and a parameterized constructor to take in all attributes in the given order:   
   **User ( String name, String emailId, String contactNumber )**
4. When the “User” object is printed, it should display the following details: **[Override the toString method]**  
   Print format:  
   Name:"name"  
   Email Id:"emailId"  
   Contact Number:"contactNumber"
5. Two Users are considered same if they have the same name(Case – Insensitive), and emailId. Implement the logic in the appropriate function. **[Override the equals method]**  
     
   The input format consists of User details separated by comma in the below order,  
   name, emailId, contactNumber

The Input to your program would be details of two Users, you need to display their details as given in "5th point(refer above)" and compare the two Users and display if the Users are same or different.  
  
**Note:**There is an empty line between display statements. Print the empty lines in main function.  
              Display one digit after the decimal point for Double datatype.  
  
  
**Sample Input & Output 1:**  
  
Enter user 1 details:  
**Slade,slade@gmail.com,9584758652**  
Enter user 2 details:  
**Slade,slade@gmail.com,9584758652**  
  
User 1  
Name:Slade  
Email Id:slade@gmail.com  
Contact Number:9584758652  
  
User 2  
Name:Slade  
Email Id:slade@gmail.com  
Contact Number:9584758652  
  
User 1 is same as User 2  
  
**Sample Input & Output 2:**  
  
Enter user 1 details:  
**Slade,slade@gmail.com,9584758652**  
Enter user 2 details:  
**Winn,winn@gmail.com,9231252136**  
  
User 1  
Name:Slade  
Email Id:slade@gmail.com  
Contact Number:9584758652  
  
User 2  
Name:Winn  
Email Id:winn@gmail.com  
Contact Number:9231252136  
  
User 1 and User 2 are different

**Learning Platform - Requirement 2**

**Requirement 2:**  
   In this requirement, you need to validate the contact number of the user.  
  
a)Create a Class **Main** with the following static methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| static Boolean validateContactNumber(String contactNumber) | Validate the contact number based on the rules given below. Returns **true**ifcontactNumber is valid else return **false** |

b) While validating contact number follow the below rules,  
  
1.The contact number should have prefix as **"+91"**(international calling code).  
2.The contact number should have 10 digit numbers [0-9].  
3.First digit of the contact number should be greater than zero.  
  
**Note:** Print "**Contact number is valid**" if mobile number is valid else print "**Contact number is invalid**".  
            All the above print statements are present in the main method.  
  
**Sample Input and Output 1:**  
  
Enter the contact number to be validated:  
**+918876543210**  
Contact number is valid  
  
**Sample Input and Output 2:**  
  
Enter the contact number to be validated:  
**+9187465123**  
Contact number is invalid

**Learning Platform - Requirement 3**

**Requirement 3:**  
In this requirement develop a feature in which you can search a List of Courses by name, and category.  
  
a) Create a Class **Course** with the following attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| Name | String |
| Category | String |
| Price | Double |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Course( String name, String category, Double price )**  
  
b) Create a class **CourseBO**with the following methods,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| public List<Course> findCourse(List<Course> courseList,List<String> nameList) | This method accepts a list of courses and list of name attributes as arguments and returns a list of courses that matches with the given list of name attributes. The returned course list should be in the order of the given nameList. |
| public List<Course> findCourse(List<Course> courseList,String category) | This method accepts a list of courses and category as arguments and returns a list of courses that matches with the given category. |

The course details should be given as a comma-separated value in the below order,  
**name, category, price**  
  
When the “course” object is printed, it should display the following details  
Print format:  
**System.out.format("%-15s %-15s %s\n","Name","Category","Price");**  
  
**Note:**The course lists are displayed in the main method.  
            If any other choice is selected, display "**Invalid Choice**"  
            If the search detail is not found, display "No such course is present"  
            Display one digit after the decimal point for Double Datatype.  
  
The above statements are displayed in the main method.  
**Sample Input and Output 1:**  
  
Enter the number of courses:  
**5  
ANGULAR,Web Development,9360  
Web Developer,Web Development,9600  
Java,Programming,4999  
Music Theory,Music,9499  
PhotoShop CS,Design,9299**  
Enter a search type:  
1.By Category  
2.By List of Names  
**1**  
Enter the Category:  
**Web Development**  
Name            Category        Price  
ANGULAR         Web Development 9360.0  
Web Developer   Web Development 9600.0  
  
**Sample Input and Output 2:**  
  
Entert the number of courses:  
**5  
ANGULAR,Web Development,9360  
Web Developer,Web Development,9600  
Java,Programming,4999  
Music Theory,Music,9499  
PhotoShop CS,Design,9299**  
Enter a search type:  
1.By Category  
2.By List of Names  
**2**  
Enter the Names:  
**Music Theory,Java**  
Name            Category        Price  
Music Theory    Music           9499.0  
Java            Programming     4999.0

**Learning Platform - Requirement 4**

**Requirement 4:**  
In this requirement, given a list of courses, you need to find the number of courses belonging to each category.

a) Create a Class Course with the following attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| Name | String |
| Category | String |
| Price | Double |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Course( String name, String category, Double price )**  
  
b) Create the following static method in the **Course** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| Map<String,Integer> coursePerCategory(List<Course> courseList) | This method takes course list as argument and returns a tree map with category as key and count of course in the category as value |

In tree map the category shoould be key and the count of courses in each category as respective value.  
  
The course details should be given as a comma-separated value in the below order,  
**name, category, price  
  
Print format:  
System.out.format("%-20s %s\n","Category","Count");**  
  
**Sample Input and Output 1:**  
  
Enter the number of courses:  
**5  
Become a Singer,Music,7500  
TShirt Design,Design,8999  
AWS Developer,IT Software,10000  
HTML CSS,Web Development,9000  
MEAN Stack,Web Development,9599**  
Category             Count  
Design               1  
IT Software          1  
Music                1  
Web Development      2

**Learning Platform - Requirement 5**

**Requirement 5:**  
  
In this requirement, you need to sort the list of courses based on no.of users and price.  
  
Create a Class **User** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| Name | String |
| emailId | String |
| contactNumber | String |
| userScoreList | List<UserScore> |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:  
**User(String name, String emailId, String contactNumber,List<UserScore> userScoreList)**  
  
The following methods are present in the **User** class

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| static List<User> prefill() | This method returns a List of prefilled User objects (Given in the template) |

Create a Class **Course** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| Name | String |
| Category | String |
| Price | Double |
| userScoreList | List<UserScore> |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:  
**Course(String name, String category, Double price,List<UserScore> userScoreList)**  
  
The **course** details are given in comma-separated values as given below,  
**name,category,price**  
  
Create a Class **UserScore** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| completionPercentage | Double |
| User | User |
| Course | Course |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:  
**UserScore(Double completionPercentage, User user, Course course)**  
  
The **UserScore** details are given in comma-separated values as given below,  
**completionPercentage, userName**  
  
The **Course** class should implement the **Comparable** interface which sorts the Course list based on no.of Users.While comparing, no.of users in the list are unique. (The number of users of a course can be computed from the userScoreList attribute)  
  
Create a class PriceComparator which implements **Comparator** interface and sort the Course list based on price. While comparing, all the price attributes in the list are unique.  
  
Get the number of Courses and course details and create an course list. Sort the courses according to the given option and display the list.  
  
Print Format: **System.out.format("%-15s %-15s %-8s %s\n","Name","Category","Price","No.of Users");  
  
Note:**Double values are displayed with one decimal place.  
                   use the prefill() method in user class to find the user object.  
                   
**Sample Input and Output:**  
  
Enter number of courses:  
**3**  
Enter course 1 detail  
**Java,Programming,4999**  
Enter number of user scores:  
**2  
95.2,Harry  
94.3,Oliver**  
Enter course 2 detail  
**Python,Programming,5299**  
Enter number of user scores:  
**1  
96.5,Harry**  
Enter course 3 detail  
**C Sharp,Programming,4900**  
Enter number of user scores:  
**3  
78.6,Matt  
60.2,James  
45.3,Harry**  
Enter a type to sort:  
1.Sort by No.of Users  
2.Sort by Price  
**1**  
Name            Category                 Price       No.of Users  
Python           Programming     5299.0    1  
Java                Programming     4999.0    2  
C Sharp        Programming     4900.0    3  
---------------------------------------------------------------------

**Learning Platform - Requirement 6**

**Requirement 6:**  
  
In this requirement, let's find the top performer in each courses from the given user scores.  
  
Create a Class **User** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| Name | String |
| emailId | String |
| contactNumber | String |
| userScoreList | List<UserScore> |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:   
**User(String name, String emailId, String contactNumber,List<UserScore> userScoreList)**  
  
The Following methods are present in the **User** class

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| static List<User> prefill() | This method returns a List of prefilled User objects (Given in the template) |

Create a Class **Course** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| Name | String |
| Category | String |
| Price | Double |
| userScoreList | List<UserScore> |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:   
**Course(String name, String category, Double price,List<UserScore> userScoreList)**  
  
The Following methods are present in the **Course** class

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| static List<Course> prefill() | This method returns a List of prefilled Course objects (Given in the template) |

Create a Class **UserScore** with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| completionPercentage | Double |
| user | User |
| course | Course |

Create / Generate appropriate Getters & Setters,  
Add a default constructor and a parameterized constructor to take in all attributes in the given order:   
**UserScore(Double completionPercentage, User user, Course course)**  
  
The Following methods are present in the **UserScore** class

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| static Map<String,UserScore> getTopPerformer(List<UserScore> scoreList) | This method returns a Map with Course name as Key and Top performer's UserScore object as Value in alphabetical order of the course name. |

**Note**: The Top performer of a course is an user with maximum **completionPercentage** in the course.  
          Use **TreeMap** in the getTopPerformer method.  
               Use **System.out.format("%-15s %s\n","Course","Top performer")**

**Sample Input/Output 1:**

Enter the number of user scores:

**6**

**95.2,Harry,JavaScript**

**94.3,Oliver,HTML CSS**

**86.8,James,Ethical Hacking**

**90.0,Danny,Ethical Hacking**

**79.6,Danny,PhotoShop CS**

**87.3,Oliver,JavaScript**

Course                Top performer

Ethical Hacking   Danny

HTML CSS          Oliver

JavaScript           Harry

PhotoShop CS    Danny

**Sample Input/Output 2:**

Enter the number of user scores:

**7**

**39.8,Danny,MEAN Stack**

**66.2,Danny,AWS Developer**

**96.5,Harry,Java**

**56.9,Brandon,Java**

**78.4,Brandon,Web Developer**

**93.5,Barry,Web Developer**

**56.3,Winn,MEAN Stack**

Course                 Top performer

AWS Developer   Danny

Java                     Harry

MEAN Stack       Winn

Web Developer   Barry

**//User prefill**

**public static List<User> prefill(){**

**List<User> list = new ArrayList<>();**

**list.add(new User("Harry","harry@gmail.com","9874585258",new ArrayList<UserScore>()));**

**list.add(new User("Oliver","oliver@gmail.com","9515951263",new ArrayList<UserScore>()));**

**list.add(new User("Danny","danny@gmail.com","8745874585",new ArrayList<UserScore>()));**

**list.add(new User("Matt","matt@gmail.com","9636925686",new ArrayList<UserScore>()));**

**list.add(new User("James","james@gmail.com","8454585263",new ArrayList<UserScore>()));**

**list.add(new User("Rob","rob@gmail.com","8475216953",new ArrayList<UserScore>()));**

**list.add(new User("Brandon","brandon@gmail.com","8457548965",new ArrayList<UserScore>()));**

**list.add(new User("Winn","winn@gmail.com","9231252136",new ArrayList<UserScore>()));**

**list.add(new User("Barry","barry@gmail.com","9568956235",new ArrayList<UserScore>()));**

**list.add(new User("Joe","joe@gmail.com","8525623568",new ArrayList<UserScore>()));**

**list.add(new User("John","john@gmail.com","8454575412",new ArrayList<UserScore>()));**

**list.add(new User("Slade","slade@gmail.com","9584758652",new ArrayList<UserScore>()));**

**return list;**

**}**

**//Course prefill**

**public static List<Course> prefill(){**

**List<Course> list = new ArrayList<>();**

**list.add(new Course("ANGULAR","Web Development",Double.parseDouble("9360"),new ArrayList<UserScore>()));**

**list.add(new Course("Web Developer","Web Development",Double.parseDouble("9600"),new ArrayList<UserScore>()));**

**list.add(new Course("JavaScript","Web Development",Double.parseDouble("9699"),new ArrayList<UserScore>()));**

**list.add(new Course("HTML CSS","Web Development",Double.parseDouble("9000"),new ArrayList<UserScore>()));**

**list.add(new Course("MEAN Stack","Web Development",Double.parseDouble("9599"),new ArrayList<UserScore>()));**

**list.add(new Course("C","Programming",Double.parseDouble("5000"),new ArrayList<UserScore>()));**

**list.add(new Course("Java","Programming",Double.parseDouble("4999"),new ArrayList<UserScore>()));**

**list.add(new Course("Python","Programming",Double.parseDouble("5299"),new ArrayList<UserScore>()));**

**list.add(new Course("C Sharp","Programming",Double.parseDouble("4900"),new ArrayList<UserScore>()));**

**list.add(new Course("AWS Developer","IT Software",Double.parseDouble("10000"),new ArrayList<UserScore>()));**

**list.add(new Course("Ethical Hacking","IT Software",Double.parseDouble("9300"),new ArrayList<UserScore>()));**

**list.add(new Course("PhotoShop CS","Design",Double.parseDouble("9299"),new ArrayList<UserScore>()));**

**list.add(new Course("TShirt Design","Design",Double.parseDouble("8999"),new ArrayList<UserScore>()));**

**list.add(new Course("3D Modelling","Design",Double.parseDouble("11999"),new ArrayList<UserScore>()));**

**list.add(new Course("Music Theory","Music",Double.parseDouble("9499"),new ArrayList<UserScore>()));**

**list.add(new Course("Learn Guitar","Music",Double.parseDouble("9999"),new ArrayList<UserScore>()));**

**list.add(new Course("Become a Singer","Music",Double.parseDouble("7500"),new ArrayList<UserScore>()));**

**return list;**

**}**