**Classes and Attributes**

Create a class named **Country**with the following public attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| name | String |
| countryCode | String |
| isdCode | String |

Create a driver class called **Main**. In the main method, obtain input from the user in the console and create objects for the class Country class using the constructor.

**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
**Input and Output format :**  
Refer to sample Input and Output for formatting specifications.  
  
**Sample Input and Output :**  
**[All Texts in bold corresponds to the input and rest are output]**  
  
Enter the country name  
**India**  
Enter the country code  
**IND**  
Enter the isd code  
**91**  
Country Name : India  
Country Code : IND  
ISD Code : 91

|  |
| --- |
| package com.company; import java.lang.\*; import java.util.Scanner;  public class Main {   public static void main(String[] args) {  String name;  String country\_code;  String isd\_code;  Scanner sc = new Scanner(System.*in*);  System.*out*.println("Enter the Country name");  name = sc.nextLine();  System.*out*.println("Enter the Country code");  country\_code = sc.nextLine();  System.*out*.println("Enter the isdcode");  isd\_code = sc.nextLine();  System.*out*.println(" Country Name : " + name);  System.*out*.println(" Country Code : " + country\_code);  System.*out*.println(" ISD Code: " + isd\_code);  } } |
|  |

**Types of Constructors**

Now you know to create an object and use getters and setters. Next, we move on to another method of object creation, using Constructors. We are going to use the default and parameterized constructor here.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class named **StallCategory** with following private attributes,

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| name | String |
| detail | String |

Include appropriate getters and setters for the attributes of the **StallCategory** class.  
  
Include the following methods in the **StallCategory**class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| StallCategory() | Default constructor to initialize the default values |
| StallCategory(String name, String detail) | Parameterized constructor to assign values to its attributes in the order. |
| void display() | This method used to display the details of the StallCategory as per the specification provided in the Sample Input and Output |

Create a driver class called **Main**. In the main method, obtain input from the user in the console and create objects for the class StallCategory class using both the constructor and find that there is no difference between them.  
  
**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.  
  
**Sample Input and Output 1:**  
**[All Texts in bold corresponds to the input and rest are output]**  
  
Enter the name of the stall category:  
**Book**  
Enter the details of the stall category:  
**All latest books are available under this category**  
Using Default Constructor  
Details of the stall category:  
Name:Book  
Detail:All latest books are available under this category  
Using Parameterised Constructor  
Details of the stall category:  
Name:Book  
Detail:All latest books are available under this category

**Overriding toString()**

Your application should have a provision to add a user by entering his/her details. The user details include username, contact detail, etc. The address of the user should also be included. Yet address may consist of several attributes. Hence create a console application for entering the address for a user.

**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a class named **Address** with the following private attributes.

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| addressLine1 | String |
| addressLine2 | String |
| city | String |
| state | String |
| pincode | Integer |

Include appropriate default constructor, getters and setters for the attributes of the **Address**class.  
Include a parameterized Constructor for the **Address**class with all attributes. Have a look at the below example  
**Address(String addressLine1, String addressLine2, String city,String state, Integer pincode)**

Override the **toString()**method in the **Address** class to display the address as per the specifications specified in the Sample Input and Output.  
  
Create a driver class called **Main**. In the Main method, obtain input from the console and create a new object for the class Address. Display the address details with the help of the Address object.  
  
**Note:** The print statement "**The Address Details are**" should be in the Main class.  
  
**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.  
  
**Sample Input and Output:**  
**[All text in bold corresponds to input and rest corresponds to ouput]**  
  
Enter Address Line 1  
**22nd Lane**  
Enter Address Line 2  
**8th cross road, RR nagar**  
Enter the City Name  
**Chennai**  
Enter the State Name  
**Tamilnadu**  
Enter the Pincode  
**620001**  
The Address Details are  
22nd Lane  
8th cross road, RR nagar  
Chennai  
Tamilnadu  
620001

**BO Classes**

Now for doing any computation works with the class its not advised to use POJO class. So we can use a BO class for computational purposes.  
  
The Stall owners wanted to calculate the total cost of a particular ItemType for the given timeline. To add a feature in the application, calculate the total cost for the given timeline.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class named **ItemType**with the following private attributes,

|  |  |
| --- | --- |
| **Attribute** | **Data Type** |
| name | String |
| deposit | Double |
| costPerDay | Double |

Add appropriate getter/setter, default and parameterized constructor.  
**public ItemType(String name, Double deposit, Double costPerDay).**

Create a class named **ItemTypeBO**and include the following method in the class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public Double calculateCost(Date start,Date end,ItemType typeIns) | This method is used to calculate the total cost. It contains start date, end date, item type object as the arguments. It will return a Double which corresponds to the total cost. |

Create a driver class **Main**to test the above classes. Get the start date and end date (manipulate as Date object) from the stall owners to calculate rent for the particular ItemType.  
  
**Input format:**  
The first line of input is a String which corresponds to the Item type name.  
The next two lines of inputs are the double which corresponds to the deposit and the cost per day.  
The next two lines of inputs are the start date **(dd-MM-yyyy)** and end date **(dd-MM-yyyy).**  
Refer to Sample Input and Output for formatting specifications.  
  
**Output format:**  
Display the value of the total cost correct to 1 decimal place.  
Refer to Sample Input and Output for formatting specifications.  
  
**Sample Input and Output:**  
**[All text in bold corresponds to the input and rest corresponds to output]**  
  
**Sample Input and Output 1:**  
  
Enter the details of the item type  
Name:  
**Bench**  
Deposit:  
**1000.00**  
Cost per Day:  
**50.00**  
Enter the starting date:  
**12-12-2017**  
Enter the ending date:  
**14-12-2017**  
The total cost is Rs.100.0  
  
**Sample Input and Output 2:**  
  
Enter the details of the item type  
Name:  
**Rolling Fan**  
Deposit:  
**500.00**  
Cost per Day:  
**50.00**  
Enter the starting date:  
**29-12-2017**  
Enter the ending date:  
**02-01-2018**  
The total cost is Rs.200.0

**Array of Objects - 2**

Yeah! Now we have completed creating and searching an array of objects. But it won't be enough for an application. So we must learn to further manipulate the array of objects. Here we will add the feature of sorting and deleting an object from the array.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class named **User** with following private attribute,

|  |  |
| --- | --- |
| **Attribute** | **Data Type** |
| name | String |
| mobileNumber | String |
| username | String |
| password | String |

Include appropriate getters/setters method, default constructor, and parameterized constructor  
**public User(String name, String mobileNumber, String username, String password)**.  
  
Now create a class named **UserBO** for manipulating user objects, create an array for storing user objects in UserBO class.

Include the following methods in the **UserBO** class

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void addUser(User[] userArrray,User userIns) | The method accepts an array of user objects and the new user object to be added as the arguments. This method adds the new user object to the array. |
| public void sortUsers(User[] userArrray) | The method accepts an array of user objects as an argument. This method sort the array of user objects based on the name |
| public Boolean deleteUser(User[] userArrray,String name) | The method accepts 2 arguments. The first argument is an array of user objects and the name of the user. This method deletes the specific user from the array and returns true if the user presents. Otherwise returns false. |
| public void displayAll(User[] userArray) | The method accepts an array of user objects. This method used to displays all the user details |

Create a driver class called **Main** to test the above classes.  
  
**Input and Output format:**  
If the user to be deleted is not present print "**No user found with given name**" else print "**User deleted successfully**" in the main method.  
Refer to Sample Input and Output for formatting specifications.  
  
**Sample Input and Output:**  
**[All text in bold corresponds to the input and rest corresponds to output]**  
  
Enter the number of users:  
**2**  
Enter the details of User 1  
Enter the name of the user:  
**John**  
Enter the mobile number of the user:  
**12345**  
Enter the username of the user:  
**john**  
Enter the password of the user:  
**john**  
Enter the details of User 2  
Enter the name of the user:  
**Joe**  
Enter the mobile number of the user:  
**67890**  
Enter the username of the user:  
**joe**  
Enter the password of the user:  
**joe**  
User details as entered:  
User Details:  
User 1  
Name:John  
Mobile Number:12345  
User 2  
Name:Joe  
Mobile Number:67890  
After sorting:  
User Details:  
User 1  
Name:Joe  
Mobile Number:67890  
User 2  
Name:John  
Mobile Number:12345  
Enter the user to be deleted:  
**John**  
User deleted successfully   
After Deleting:  
User Details:  
User 1  
Name:Joe  
Mobile Number:67890

**User Credentials**

In a typical application, the password will not be stored directly in the database as it will be more prone to attack. The password of users will always be stored in an encrypted form. To make your application secure, the password has to be in an encrypted format. Hence implement a typical user authorization technique for login.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement**  
  
Create a class named **User** class with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| id | Integer |
| username | String |
| password | String |

Include appropriate getters and setters  
Create Default Constructor and Parameterized Constructor as **User(Integer id, String username, String password)** for the class.  
  
Create a class named **UserBO** class to validate the username and password with the following methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| User[] getUsers() | This method returns a predefined array of users. |
| private String encryptPassword(String password) | This method accepts a single argument of type String. It returns the encrypted password by replacing it with its next ASCII character. **For example**, A is replaced with B. |
| boolean validate(String username, String password) | This method takes username and password entered by the user as arguments. The password has to be encrypted. The username and the encrypted password has to be compared with the predefined user list. It returns boolean after validating. |

Create a driver class named Main to test the above class.  
  
**Sample user list:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Username** | **Password** |
| 1 | Louis | rxfsuzA2345 |
| 2 | Messie | hpphmf1: |
| 3 | Steve | opefKt |
| 4 | Kallis | 23456778 |
| 5 | Wipro | A$% |

**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.

**Note: All Texts in bold corresponds to the input and rest are output  
  
Sample Input and Output 1:**  
Username :  
**Louis**  
Password :  
**qwerty@1234**  
Login Successfull  
      
**Sample Input and Output 2:**  
Username :      
**Messie**  
Password :  
**google**  
Incorrect username/password

**Big Bash Event**

The fair has an event called the Big Bash event. It is conducted to increase the business of the stalls. It gives a discount on the particular bills and the constraint is not told to the audience attending the fair. Create a program to check whether a bill is eligible for the BigBash event or not. The eligibility is calculated on the basis of the purchased date. If the month in the purchased date is even, then the bill is eligible for the event. If the purchased month is odd, then it is not eligible for the event. If the bill is eligible for the event, then the discount is given. The discount percentage should be the purchased month number.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
**Example:**  
If purchased date is --> 12-10-2017 [**dd-MM-yyyy**]  
The purchased month is 10, so Peter is eligible for the event and a discount of 10% should be given to the user.  
If the purchased amount is 100, then the discount amount is 10. So, the total amount is 100-10= 90.  
If purchased date is --> 12-01-2018 [**dd-MM-yyyy**]  
The purchased month is 01, so Peter is not eligible for the event.  
  
Create a class named **Event** with the following private methods,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Object checkEventAvailable(String date) | This method takes the date of purchase as a string and checks for the month. If the month is even it should create a date object(java.util.Date) from the String. It returns the created date object.Otherwise, it returns the String "Not Eligible for BIGBASH event" |
| static Double getAmountWithDiscount(Double amount, Date date) | This method takes the purchase amount and date of purchase as arguments. calculate the amount of discount and return the discounted amount as Double. |

Create a driver class **Main** to test the above class.  
  
**Problem Constraints:**  
1. Use **instanceof** operator to check the object returned by checkEventAvailable method.  
**[**The java **instanceof**operator is used to test whether the object is an instance of the specified type (class or subclass or interface).**]**  
  
**Input format:**  
The first line of the input is a String which corresponds to the purchased date.  
The second line of input is an integer which corresponds to the purchase amount **[only if he is eligible for the event].**  
  
**Output format:**  
If he is eligible for the event, the discount amount has to be displayed. Otherwise, display "**Not Eligible for BIGBASH event**".  
Refer to sample output for formatting specifications.  
  
**Note: All text in bold corresponds to input and rest corresponds to output.**  
  
**Sample Input and Output 1:**  
Enter the purchased date:  
**12-12-2017**  
Enter purchase amount:  
**100**  
The discounted amount is 88.0  
  
**Sample Input/Output 2:**  
Enter the purchased date:  
**12-01-2018**  
Not Eligible for BIGBASH event

**User Matcher**

Previously we had implemented User Authorization in our application. We had forgotten to implement to check duplicate user details. In our application, there can be only one user with a specific mobile number. If two users exist with the same mobile number they are duplicate. How to check whether two objects are equal? Yeah, you are right the **equals()** method. Go on override the equals method and check if two users are the same.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class named **User**with following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| username | String |
| password | String |
| mobileNumber | Long |

Include appropriate getters and setters.  
Create Default Constructor and Parameterized Constructor as **User(String name, String username, String password, Long mobileNumber)** for the class.

**Problem Constraints:**

Override**equals method** that compares mobileNumber of the two User objects.   
If the mobileNumber of the two User is the same, then it has to be print “**User 1 and User 2 are equal**”. Otherwise, it has to be print “**User 1 and User 2 are not equal”.**  
  
Create a driver class named **Main** to test the above class.  
  
**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.  
  
**Note:** **All text in bold corresponds to input and rest corresponds to output.**  
  
**Sample Input and Output 1:**  
Enter the details of User 1  
Name :  
**Morsh**  
Username :  
**Morsh2020**  
Password :  
**Basic**  
Mobile Number :  
**1234567890**  
Enter the details of User 2  
Name :  
**Morsh**  
Username :  
**Morsh1010**  
Password :  
**Probob**  
Mobile Number :  
**0987654321**  
User 1 and User 2 are not equal  
  
**Sample Input and Output 2:**  
Enter the details of User 1  
Name :  
**Bajaj**  
Username :  
**Ba450**  
Password :  
**particles**  
Mobile Number :  
**9894098490**  
Enter the details of User 2  
Name :  
**Bajaj**  
Username :  
**Bajji**  
Password :  
**500**  
Mobile Number :  
**9894098490**  
User 1 and User 2 are equal

**Array of ItemType objects**

Kudos, your application is progressing really good but its time to add some really cool features to it. Until now you have created single objects for each class, now there's a need to create an array of objects for a class to enable the search feature. Create an array of ItemType objects. Also, add a provision for adding a new ItemType object to the array of ItemType objects. Create an interactive console to get input from the user for performing different actions on the array.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class named **ItemType** with the following private attributes.

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| deposit | Double |
| costPerDay | Double |

Include appropriate getters and setters for the class.  
Include default and parameterized constructors for the class.  
  
Create a class named **ItemTypeBO** class and include the following methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(ItemType object ,ItemType[] itemTypeArray,Integer index) | This method accepts an array of ItemType objects, an ItemType object and the index at which the object is to be added. The method appends the object to the array of ItemType objects at the specified index and prints "**New item added successfully**" |
| public void search(String search, ItemType[] itemTypeArray) | This method accepts the search term and the array of ItemType objects as input. It displays the details of the ItemType if matched. else prints "**Searched Item Type not found**" |
| public void display(ItemType[] itemTypeArray) | This method accepts an array of ItemType objects as arguments. It displays the details one by one. |

Create a driver class called **Main**. In the Main method, obtain input from the user and call the ItemTypeBO functions appropriately.  
  
**Input and Output format:**  
The size of the itemType Array should be 10. (1<=n>=10).  
Display the value of the deposit amount correct to 1 decimal place.  
The search statement "**Enter the Name of the item to be searched**" should be in the Main method.  
Refer to sample Input and Output for formatting specifications.  
  
  
**Sample Input and Output 1:**  
**[All text in bold corresponds to the input and rest corresponds to output]**  
  
Enter the Number of Item Type  
**3**  
Enter the Item Type 1 Name  
**Electronics**  
Enter the Deposit Amount  
**2500**  
Enter the Cost per day  
**150**  
New item added successfully  
Enter the Item Type 2 Name  
**Chemicals**  
Enter the Deposit Amount  
**500**  
Enter the Cost per day  
**50**  
New item added successfully  
Enter the Item Type 3 Name  
**Construction**  
Enter the Deposit Amount  
**2000**  
Enter the Cost per day  
**500**  
New item added successfully  
Item Type Number 1:  
Name:Electronics  
Deposit Amount:2500.0  
Cost Per Day:150.0  
Item Type Number 2:  
Name:Chemicals  
Deposit Amount:500.0  
Cost Per Day:50.0  
Item Type Number 3:  
Name:Construction  
Deposit Amount:2000.0  
Cost Per Day:500.0  
Enter the Name of the item to be searched  
**Construction**  
Searched Item Type is:  
Name:Construction  
Deposit Amount:2000.0  
Cost Per Day:500.0

**Sample Input and Output 2:**  
Enter the Number of Item Type  
**2**  
Enter the Item Type 1 Name  
**Electricals**  
Enter the Deposit Amount  
**2000**  
Enter the Cost per day  
**200**  
New item added successfully  
Enter the Item Type 2 Name  
**Mechanic kit**  
Enter the Deposit Amount  
**3000**  
Enter the Cost per day  
**250**  
New item added successfully  
Item Type Number 1:  
Name:Electricals  
Deposit Amount:2000.0  
Cost Per Day:200.0  
Item Type Number 2:  
Name:Mechanic kit  
Deposit Amount:3000.0  
Cost Per Day:250.0  
Enter the Name of the item to be searched  
**Electronics**  
Searched Item Type not found

**Hall Equality**

Now we know how to use the equals() method. Now let's use it in our application to reduce redundancy. The same halls cannot be booked on the same date for two different events. So get the halls booked by two different persons and check whether it is the same or not. Use equals method to check all attributes of the hall are equal including booking date.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement**  
  
Create a class **Hall**with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| contactNumber | Long |
| costPerDay | Double |
| ownerName | String |
| bookingDate | Date |

Include appropriate getters and setters  
Create Default and Parameterized Constructor as **Hall(String name, Long contactNumber, Double costPerDay, String ownerName, Date bookingDate)** for the class.  
  
Override **equals method** that compares all the fields of the class and returns boolean.  
  
Create a driver class named **Main** to test the above class.  
  
If all the fields are the same then print “**Halls are same**”. Otherwise, print “**Halls are different**” in the Main.  
  
**Input and Output format:**  
Refer to sample Input and Output for formatting specifications.  
  
**Note:**  **All Texts in bold corresponds to the input and rest are output**  
  
**Sample Input and Output 1:**  
  
Enter the details of Hall 1  
Name :  
**Platinum**  
Contact Number :  
**2200889**  
Cost Per Day :  
**32000**  
Owner Name :  
**Bala**  
Booking Date(dd/mm/yyyy) :  
**12/12/2017**  
Enter the details of Hall 2  
Name :  
**Platinum**  
Contact Number :  
**2090909**  
Cost Per Day :  
**48000**  
Owner Name :  
**Krishna**  
Booking Date(dd/mm/yyyy) :  
**12/12/2017**  
Halls are different  
  
**Sample Input and Output 2:**       
  
Enter the details of Hall 1  
Name :  
**Euro**  
Contact Number :  
**224400**  
Cost Per Day :  
**1200**  
Owner Name :  
**Krish**  
Booking Date(dd/mm/yyyy) :  
**20/01/2018**  
Enter the details of Hall 2  
Name :  
**Euro**  
Contact Number :  
**224400**  
Cost Per Day :  
**1200**  
Owner Name :  
**Krish**  
Booking Date(dd/mm/yyyy) :  
**20/01/2018**  
Halls are same