Core Java Exercises

1. Create an Entity class '**Trainee**', with traineeId, traineeName, contactNo, email, gender, age as fields.

   Create a class 'Batch', with batchCode, startdate, enddate and Trainee[] as fields.
   Create the following overloaded methods in the 'Batch' class

   public Trainee getTrainee(int traineeId) thows TraineeNotFoundException
   public Trainee[] getTrainees(String gender)
   public Trainee[] getTrainees(int age)

2. Follow the given instructions and create an application using Java.
   - (i) Create an entity class named Project with member variables as projectId, projectName, projectHead, noOfResources.
   - (ii) Create an object for the Project class and through setters assign the values for all the member variables.
   - (iii) Print the corresponding object.

3. Prepare a **StringServiceProvider** class which has the following methods
   - (a) To reverse a given string
   - (b) To do linear search in a given string
   - (c) To do search and replace operation in a given string

   Note: code the requirement with 2 possibilities (with and without static methods)

4. Follow the given instructions and create an application using Java.
   - (i) Create a '**BankAccount**' class with 3 data members, accountNo[use String], accountName and balance.
   - (ii) Overload the BankAccount constructor to accept only accountNo and accountName variables.
   - (iii) Initialize the balance with 1000.

5. Follow the given instructions and create an application using Java.

   (i) Create a Bank Class having an array of BankAccount as a data member, populate the array through setters.

   (ii) Introduce a static variable called lastAssignedNo(integer). This should be initialized to 0 in the beginning. While creating new bank accounts, the accountNo variable should not be supplied in the constructor parameter list. Instead it has to be computed as (lastAssignedNo + 1). Also modify the lastAssignedNo after creating a bank account.

(iii) Create an **IBankServiceProvider** interface and declare the following methods:

(a) **BankAccount checkAccount(String accountNo)**
This checks whether the given account number is available in the array or not. If exists, it should return the object of BankAccount class, else return null. *Reuse this method in all the other methods given below*.

(b) **double getBalance(String accountNo)**
This will return the balance in an account for the given account

(c) **boolean depositMoney(String accountNo, double amount)**
This deposits the given amount into the given account number after verifying whether the given account is present in the array or not.

(d) **boolean withdrawMoney(String accountNo, double amount)**
This will withdraw the given amount from the given account after verifying the existence of account as well as balance.

(e) **boolean transferMoney(String fromAccountNo, String toAccountNo, double amount)**
This transfers the money from one account to another account after verifying both the accounts are existing or not as well as balance of the 'fromAccount'.

(iv) Bank class should implement **IBankServiceProvider** interface and override the methods of the interface.

6. Create user defined exceptions, **InsufficientFundException** and **InvalidAmountException** classes.

   Throw these exceptions from the methods in Bank class. Make necessary changes to accommodate these exceptions in the source code. Handle all these exceptions from the main program.

7. Design a class **Company** which has the following attributes:
   Company ID
   Address
   EmployeeMap < id, Employee>

   → Employee is an Entity class
   → Map<Key, Value> is EmployeeMap<empID, Employee>
   → **Company** class should implement **ICompanyserviceprovider** to facilitate basic CRUD operations on Employee Class.
   (Consider using a List over the EmployeeMap and update the necessary changes in the code)

8. Understand the below given requirements and design the system with the required Entity classes that has data members, properties and member functions.

   Requirements:
   The proposed system will follow the general operations of the library like, maintaining different kinds of books categorized as fiction, novel and general by

giving a unique book id and recording the book name, author name, publisher name and the number of copies of that book.

Maintain the members who are allowed to take books from the library by giving a unique member id to each and every member and recording the member name, address and telephone number. Each member is given 3 cards, where one book can be taken on each card, and the user has to return the book within 7 days after the issue of the book, otherwise a fine of Rs. 2/- will be charged per day.

This system should generate different kind of reports like, displaying the total no. of members, displaying available books of different categories, books due on the current date, and books issued to members.

System has to take care of the following details.

- ➢ Maintaining Member details
- ➢ Maintaining Books details
- ➢ Transactions (Issue And Return of Book)
- ➢ Reports

9. Create a class Employee with following fields
   employeeID, employeeName, password, salary, deptNO
   Create a unique collection of employees and count them based on the department number and list the result. Serialize the above collection of employees into a file. While serializing rotate the password by a number, while deserializing rotate it in reverse, using the same number.

10. Write a class **Product**, with following data members and member functions for accomplishing the given requirements.:
    Fields:
    productId, productName, price, quantityOnHand, reorderLevel, reorderQty

    Assume reorderLevel of the product is 10 qty's and reorderQuantity is 50 quantities.

    Write a class **Stores** which has a List of Products as its only field. Populate the list in the constructor of Stores. Add the following methods to this class.

    double sellItem(int productCode, int qtyRequired) throws ProductNotFoundException:
It should sell the required qty and show the total amount.
This method also checks whether reorderLevel is reached. If yes, raise a purchase order.
[Just print a message on the screen saying that 'purchase order is made'].
**void updateStock(int productCode, int arrivedQty)
throws ProductNotFoundException;**

11. Create a Comparator<Product> object to sort the products in stores class with respect to price. Display the sorted products.

12. There is a class called **Pet** which is not a Serializable class.
    There is an another class called **Person** which is Serializable and has Pet as a field.
    Write a manual serialization program to serialize the Person object.