1. (Person-Student-Employee)
(The Person, Student, Employee, Faculty, and Staff classes)
Design a class named Person and its two subclasses named Student and Employee.
Make Faculty and Staff subclasses of Employee.
A person has a name, address, phone number, and e-mail address.
A student has a class status (freshman, sophomore, junior, or senior). Define the status as enum. Enum is special type used to define collections of constants.

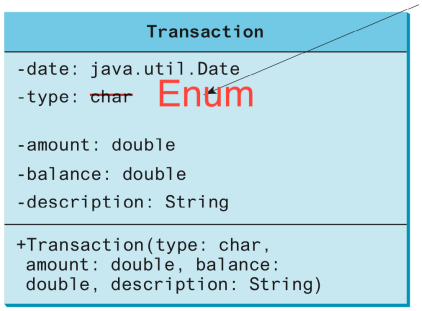
An employee has an office, salary, and date hired. Use Date class to create an object for date hired.
A faculty member has office hours and a rank.
A staff member has a title.
Override the toString() method in each class to display the class name and the person's name.
Write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.

2. (Students)
Create a class called Student.
The Student class includes a name and a Boolean value representing full-time status.
Include a method to determine the tuition, with full-time students paying a flat fee of $2,000 and part-time students paying $200 per credit hour. Create two subclasses called FullTime and PartTime. Create an application that demonstrates how to create objects of both subclasses. Save the files as Student.java, FullTime.java, PartTime.java, and UseStudent.java.

3. Account and Transaction

(*New* ***Account*** *class*) An **Account** class was specified in Programming Exercise 9.7. Design a new **Account** class as follows:

- Add a new data field **name** of the **String** type to store the name of the customer.
- Add a new constructor that constructs an account with the specified name, id, and balance.
- Add a new data field named **transactions** whose type is **ArrayList** that stores the transaction for the accounts. Each transaction is an instance of the **Transaction** class, which is defined as shown in Figure 11.6.



- Modify the **withdraw** and **deposit** methods to add a transaction to the **transactions** array list.

Write a method that returns the balance from a specific time in the past **public double getBalanceFrom(Date fromDate)**;
Write a method that returns the initial balance (regardless of how many transactions you have made)

4) ArrayList

Part 1) Write a method that sorts an ArrayList.
Method signature should be:

```
public static void sort(ArrayList<Integer> list, boolean asc, boolean builtInSort)
```

where list is an ArrayList of Integer, asc is a Boolean that signifies whether we are going to sort ascending or descending and builtInSort is another Boolean that signifies whether we want to use a built-in method of ArrayList to sort or our method.
Notice that we don't return anything, our sorting method is mutating the given list.

Part 2) Write a method that removes duplicate values from an ArrayList
Method signature should be:

```
public static ArrayList<Integer>
removeDuplicate(ArrayList<Integer> list)
```

Notice that we don't mutate the current list, but we return a new list with only unique values.

Part 3) Write a method that unions undefined number of ArrayList
Method signature should be:

```
public static ArrayList<Integer>
unionLists(ArrayList<Integer>... lists)
```

We should create a final list that unions all the lists given as arguments but only unique values.
Let's say we have A = [1, 2, 3, 4, 4], B = [4, 2, 3, 5, 6] and C = [1, 6]
Result of the method should be a new array with [1, 2, 3, 4, 5, 6] elements.

Part 4) Write a method that shuffles an ArrayList
Method signature should be

```
public static ArrayList<Integer> shuffleList
(ArrayList<Integer> list)
```

Re-organise the elements of ArrayList randomly.

5) **MyStack class**
Implement a LIFO data structure using ArrayList as a backbone
MyStack should contain a constructor that takes an array list and converts it into a LIFO and a no-args constructor.
Public methods of the class are:
isEmpty()
push()
peek()

pop()
getSize()
clear()

6) MyQueue class
Implement a FIFO data structure using ArrayList as a backbone.
MyQueue should contain a constructor that takes an array list and converts it into a FIFO and a no-args constructor.
Public methods of the class are:

isEmpty()
enqueue()
peek()
dequeue()
getSize()
clear()