### **List of Operations**

[bookmark\_border](https://doselect.com/" \o "Bookmark)

subject Coding

**DESCRIPTION**

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

**Specifications:**

class definitions:

 class ArrayListOps:

 method definitions:

  makeArrayListInt(int n): Method to **create** an arrayList **with** same number **of** elements **as** n **and** **set** elements **to** 0

**return** **type**: ArrayList<Integer>

      visibilty: **public**

﻿

  reverseList(ArrayList<Integer> **list**): Method **to** **Reverse** **list**

**return** **type**: ArrayList<Integer>

      visibilty: **public**

changeList(ArrayList<Integer> **list**, int m, int n): Method **to** **change** **all** occurences **of** m **to** n **in** **list**

**return** **type**: ArrayList<Integer>

      visibilty: **public**

**Task:**

Your task is to create a class **ArrayListOps** and implement the following:

**1.** **makeArrayListInt(int n):**Method to create an A**rrayList** with number of elements as n and set elements to **0**.

* If number of elements **n** is 4 , then the method should return a list containing **[0,0,0,0]**

**2.** **reverseList(ArrayList<Integer> list):** Method to Reverse**list**

**3. changeList(ArrayList<Integer> list, int m, int n)**: Method to change all **occurences** of **m** to **n** in **list**

**Important:**

* To check your program, you can use the **main()** method (in Source class) given in the stub. You can make suitable function calls and use **RUN CODE** button to check your main() function output.

**Sample Input**

ArrayList<Integer> list = new ArrayList<Integer>(Arrays.asList(10, 25, 33, 28, 10, 12));

n = 4(method makeArrayListInt)

m = 28, n = 20(method changeList)

**Sample Output**

[0, 0, 0, 0]

[12, 10, 28, 33, 25, 10]

[12, 10, 20, 33, 25, 10]

**NOTE:**

* The above **Sample Output** is only for demonstration purposes and will be obtained if you implement the **main()** method with all method calls accordingly.
* Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at the **Sample Output**.

**EXECUTION TIME LIMIT**

Default.

REPORT AN ISSUE

### **Handling Stuff**

[bookmark\_border](https://doselect.com/" \o "Bookmark)

* subject Coding

**DESCRIPTION**

In **Java**, we can use more than one catch block with the try block. Generally, multiple catch block is used to handle different types of exceptions, which means each catch block is used to handle different types of exceptions.

If you use multiple catch blocks for the same type of exception, then it will give you a compile-time error because **Java does not allow you to use multiple catch block for the same type of exception**. A catch block is always preceded by the try block.

Write a program to demonstrate Multiple Exceptions.

**Specifications:**

**class** **Activity**:

    data fields:

      String string1

      String string2

      String **operator**

    Constructor to initialize the **class** **variables**.

**class** **Source**:

method definitions:

handleException(Activity a): implement **try**-**catch** blocks **and** **throw** different **exceptions** **as** described **under** Tasks

**return** **type**: **String**

visibility: **public**

doOperation(Activity a): implement **switch** **statement** **to** calculate **Result** based **on** **value** **of** **Operator**

**return** **type**: **String**

visibility: **public**

You have to implement the following methods under Source class:

* **handleException (Activity a)** - In this function you have to check for exceptions.
* **doOperation (Activity a)** - this function should implement the string operation between **string1** and **string2** for the operator **operator**.
* If **operator = '+'**, concat the strings **string1** and **string2**.
* **e.g.** for **string1 = "hello"**and **string2 = "world"**, then **result** = **"helloworld"**
* If **operator = '-'**, replace the contents of **string2** in **string1** with empty string.
* **e.g.** If **string1 = "helloworld"** and **string2 = "world"**, then **result = "hello"**

**Input Format**

* The **main()** method has already been implemented, which will pass values for the variables: **string1**, **string2** and **operator**.

**Tasks:**

In the function **handleException** **(Activity a)**:

* Check that the value of either **string1** or **string2** variable is **null**, then throw appropriate exception for **NullPointerException** and return "**Null values found**".
* Check if the value of **operator** variable is not equal to these string operators ((+ or -) using logical AND operator. If the condition is true then throw and return the default exception with the Operator as the return message.
* If no exception is found return "**No Exception Found**".

In the function **doOperation (Activity a)**:

* perform the string operations, using switch statement and return the correct value.

**IMPORTANT:**

* If you want to test your program, you can implement a **main()** function given in the stub and you can use **RUN CODE** to test your main() provided you have made valid function calls with valid data required.

**EXECUTION TIME LIMIT**

### **Employee Verification**

[bookmark\_border](https://doselect.com/" \o "Bookmark)

* subject Coding

**DESCRIPTION**

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

**Specifications:**

class definitions:

  class Employee:

    data fields:

      name: String

      salary: int

    Implement a Constructor using the class variables.

    Implement getter setter methods **with** **public** visibility.

**class** EmployeeInfo:

    enum definition:

      named constants: BYNAME

                       BYSALARY

    method definitions:

**sort**(**List**<Employee> emps, **final** SortMethod method): Method **to** **return** sorted **list** **by** **name** **and** **by** salary **using** SortMethod

**Return** **type**: **List**<Employee>

           Visibility: **public**

      isCharacterPresentInAllNames(Collection<Employee> entities, **String** character): method **to** **check** **if** Employee **list** contains a **name** **starting** **with** a specific character

**Return** **type**: boolean

           Visibility: **public**

**Task:**

Create an Employee class which has the following members:

**String** name;

**int** salary;

* Define parameterized constructor.
* Define getter method for all instance variables with public visibility.(getName(),...)
* Define setter methods for all instance variables with public visibility.(setName(),....)

Create an **EmployeeInfo** class which performs following operations (as per the given requirements) using **StreamAPI**:

* **enum SortMethod** : representing a group of named constants **BYNAME** and **BYSALARY**
* **sort(List<Employee> emps, final SortMethod method)**: Method to return sorted list by **name** and by **salary** using **SortMethod**
* **isCharacterPresentInAllNames(Collection<Employee> entities, String character):** Method to check if Employee list contains a name starting with a specific character

Implement using **Lambda expressions**.

Following has been done for you:

* **Main()** method containing list of **Employees**
* **String toString()** method, it's part of code stub, don't edit it else your test-cases might fail

**Sample Input**

List<Employee> emps = **new** ArrayList<>();

emps.**add**(**new** Employee("Mickey", 100000));

emps.**add**(**new** Employee("Timmy", 50000));

emps.**add**(**new** Employee("Annny", 40000));

**Sample Output**

[<name: Annny salary: 40000>, <name: Mickey salary: 100000>, <name: Timmy salary: 50000>][<name: Annny salary: 40000>, <name: Timmy salary: 50000>, <name: Mickey salary: 100000>]false

**NOTE**

* Do not use any **for** loops or other control structures.
* Use the stream API methods for your implementations, else the test-cases might fail.
* You CAN implement the **main()** method to check the implementation of your methods in the solution.
* Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at some output.

**EXECUTION TIME LIMIT**

Default.

REPORT AN ISSUE

### **Email Operation**

[bookmark\_border](https://doselect.com/" \o "Bookmark)

* subject Coding

**DESCRIPTION**

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

class definitions:

class Header:

Variables:

String from

String to

Implement a parameterized constructor to initialize all the instance variables.

class Email:

Variables:

Header header

String body

String greetings

Implement a parameterized constructor to initialize all the instance variables.

class EmailOperations:

Methods:

emailVerify(Email e): **Use** regular expression **to** **verify** **if** the two email-ids **in** the Header **class** **is** valid **or** not.[**Return** **type** explained **in** Task part].

**Return** **type**:int

Visibility: **public**

bodyEncryption(Email e): **Use** Ceasar cipher(Shift-3) **to** **encrypt** the **body** **of** the email.[**To** know more refer the Task part]

**Return** **type**:**String**

Visibility: **public**

greetingMessage(Email e): **In** this method you have **to** **return** a greeting messgae. The greet part should be taken **from** greetings **variable** **and** signature(**name**) should be taken **from** Header's '**from**' email address.[To know more refer the Task part]

Return type:String

Visibility: public

Class Variables:

* **class Header:** It contains two email id 'from' and 'to'. 'from' signifies the sender's email address and 'to' signifies receiver's email address.
* **class Email:** This class contains three parts: first Header header which has two email address from and to,the second body which contains the message to send and third greetings which contains greetings such as "Regards", "Thank you", etc.

To access a variable in Header class through Email object we use:

<**Email(obj)**>.<**Email(variable)**>.<**Header(variable)**>

Example to access "from" address from the Email object e we use : e.header.from;

Tasks:

* Implement the two classes Email and Header class according to the specifications.
* Implement the three methods in the EmailOperations class:

1. emailVerify (Email e)
2. bodyEncryption (Email e)
3. greetingMessage (Email e)

Method Description:

1. emailVerify(Email e):

* In this method you have to use regex to check if the email-address to and from in Header class is valid or not. Validation is based on:
* Email address should start with alphabets(capital/small) or \_(underscore).
* Email address should have only one @ followed by alphabets.
* Email address should end with .(dot) followed by alphabets.
* e.g: amit@doselect.com, \_ami@doselect.in are valid addresses, but 1ami@dos.com, amit@doselect are invalid addresses.
* Return 2 if the both email addresses are valid return 1 if one is valid, and 0 if both are invalid.

2. bodyEncryption(Email e):

* In this method, you have to use Caesar cipher(shift of 3) to encrypt the body part of the Email return the encrypted string.
* Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. Here the number of shift is 3.
* e.g: str = "Hi There Hows you", after encryption becomes "Kl Wkhuh Krzv brx". H get converted to K that is a shift of 3 alphabets ahead.
* Letters which are capital should be capital and small should be small in Encrypted message. Take care of the spaces.

3. greetingMessage(Email e):

* In this method, you have to return a concatenated string which contains the greetings variable from Email class and Name of the person who is sending the mail(from variable in the Header class).
* The name part should not contain anything which is after @ in the email id.
* e.g: if greetings = "Regards" and from = "Amit@doselect.com" then you have to return the message "Regards Amit"

Important:

* To check your program you have to use the main() function(in Source class) given in the stub. You can make suitable function calls and use **RUN CODE** button to check your **main()** function output.

**EXECUTION TIME LIMIT**

Default.

REPORT AN ISSUE

import java.util.\*;

class ArrayListOps {

public ArrayList<Integer> makeArrayListInt(int n)

{ int m=n;

ArrayList<Integer> list=new ArrayList<>(m);

return list;

}

public ArrayList<Integer> reverseList(ArrayList<Integer> list)

{

ArrayList<Integer> rev=new ArrayList<Integer>();

for(int i=list.size()-1;i>=0;--i)

{

rev.add(list.get(i));

}

return rev;

}

public ArrayList<Integer> changeList(ArrayList<Integer> list , int m,int n)

{

for(int i=0;i<list.size();i++)

{

if(list.get(i)==m)

{

list.set(i,n);

}

}

return list;

}

}

public class Source{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

ArrayListOps a=new ArrayListOps();

int n=sc.nextInt();

ArrayList<Integer> arr=new ArrayList<Integer>();

ArrayList<Integer> list = new ArrayList<Integer>();

while(sc.hasNextInt())

{

int inp=sc.nextInt();

list.add(inp);

}

arr=a.makeArrayListInt(n);

System.out.println(arr);

arr=a.reverseList(list);

System.out.println(arr);

int m=sc.nextInt();

n=sc.nextInt();

arr=a.changeList(list,m,n);

System.out.println(arr);

}

}