**Creating Methods**

In this exercise, we’ll create two methods that will use the functional programming style of looping through every element in a list of numbers

**Method 1:** This method will take as input a list of integers and will return a string of odd numbers with each number in the string separated by a comma.

**Step 1:** We’ll begin by creating a method named getOddNosString that must accept a List<Integer> of nos as an input and returns back a string consisting of all the odd numbers from the input list, separated by a comma in the final string. Name the method getOddNosString and this would be a static method with public access. Create this method below the main method in the LoopNumbersListFPRunner class.  
  
public static String getOddNosString(List<Integer> nos) {

}

**Step 2:**  While looping over the numbers from the input list, we will be selectively picking only odd numbers from the list and concatenating them in the final String (along with the comma!), that needs to be returned from the method. For this purpose, we will create a StringBuilder instance

public static String getOddNosString(List<Integer> nos) {

final StringBuilder builder = new StringBuilder();

}

**Step 3:** Now, we loop through every number from the input list, using the functional programming construct `forEach` that is available to be called on the List object. It takes a lambda expression as the only argument.

public static String getOddNosString(List<Integer> nos) {

final StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

});

}

**Step 4:** Coming to the business logic, we check whether `no` (passed as an input to the lambda expression`) is an odd number or no. If it is an odd number, we append it in the StringBuilder instance created in Step 2, along with the comma!

public static String getOddNosString(List<Integer> nos) {

final StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

if (no % 2 != 0) {

builder.append(no);

builder.append(',');

}

});

}

**Step 5:** After having finished looping through each number in the list using `forEach`, check whether the StringBuilder is empty or not. It can be empty if the input list passed is empty or has only even numbers. If it is empty, directly return an empty string from here.

public static String getOddNosString(List<Integer> nos) {

final StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

if (no % 2 != 0) {

builder.append(no);

builder.append(',');

}

});

if (builder.length() == 0) {

return "";

}

}i

**Step 6:** Finally, if the StringBuilder is not empty, extract the entire concatenated String from the StringBuilder and yes, do not forget to remove the trailing comma using String manipulation.

public static String getOddNosString(List<Integer> nos) {

final StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

if (no % 2 != 0) {

builder.append(no);

builder.append(',');

}

});

if (builder.length() == 0) {

return "";

}

return builder.toString().substring(0, builder.length()-1);

}

That’s it! You’ve just built a method that can take any input list of integers and returns a string of odd numbers using functional programming construct.

**Method 2:** This method will take as input a list of integers and will return a string of numbers from the input list, but where each number is deducted by 1 in the final list and separated by a comma

**Step 1:** We’ll begin by creating a method named getDeductedMarksString that must accept a List<Integer> of nos as an input and returns back a string consisting of all the nos from the input list, but with each no. from the input list deducted by 1 and separated by a comma. Name the method getDeductedMarksString and this would be a static method with public access. Create this method below the main method in the LoopNumbersListFPRunner class.  
  
public static String getDeductedMarksString(List<Integer> nos) {

}

**Step 2:**  While looping over the numbers from the input list, we will picking the number from the list, deducting selectively by 1 and concatenating them in the final String (along with the comma!), that needs to be returned from the method. For this purpose, we will create a StringBuilder instance

public static String getDeductedMarksString(List<Integer> nos) {

StringBuilder builder = new StringBuilder();

}

**Step 3:** Now, we loop through every number from the input list, using the functional programming construct `forEach` that is available to be called on the List object. It takes a lambda expression as the only argument.

public static String getDeductedMarksString(List<Integer> nos) {

StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

});

}

**Step 4:** Coming to the business logic, we check whether `no` (passed as an input to the lambda expression`) is less than or equal to 0 or no. If it is more than 0, we deduct it by 1 and append it to the StringBuilder instance created in Step 2. If otherwise, we append the integer as is to the StringBuilder instance.

public static String getDeductedMarksString(List<Integer> nos) {

StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

builder.append(no > 0 ? no - 1 : no);

builder.append(',');

});

}

**Step 5:** After having finished looping through each number in the list using `forEach`, check whether the StringBuilder is empty or not. It can be empty if the input list passed is empty. If it is empty, directly return an empty string from here.

public static String getDeductedMarksString(List<Integer> nos) {

StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

builder.append(no > 0 ? no - 1 : no);

builder.append(',');

});

if (builder.length() == 0) {

return "";

}

}

**Step 6:** Finally, if the StringBuilder is not empty, extract the entire concatenated String from the StringBuilder and yes, do not forget to remove the trailing comma using String manipulation.

public static String getDeductedMarksString(List<Integer> nos) {

StringBuilder builder = new StringBuilder();

nos.forEach(no -> {

builder.append(no > 0 ? no - 1 : no);

builder.append(',');

});

if (builder.length() == 0) {

return "";

}

return builder.toString().substring(0, builder.length()-1);

}

That’s it! You’ve just built a method that can take any input list of integers and returns a string of integers deducted by 1 using functional programming construct.