1. Create an Interface **Palindrome** with a method *checkPalindrome()*. Construct three classes **Stack**, **StackReverse** and **QueueReverse**.

   Class **Stack** contains the following operations:
   - *push() # Push elements into the stack*
   - *pop() # Pop elements from the stack*
   - *disp() # Display the elements of the stack*

   Class **StackReverse** *(operates only on string input)* extends the class **Stack** and implements **Palindrome** with following operations:
   - *stackReverse() #Reverse the stack using the operations in Stack class*
   - *checkPalindrome() # Checks if the string in the Stack is a palindrome or not*

   Class **QueueReverse** *(operates only on digits [0-9])* extends the class **Stack** and implements **Palindrome** with the following operations:
   - *insertQueue() #Insert elements into the queue.*
   - *queueReverse() #Reverse the queue using the operations in Stack class*
   - *disp() #Display the reversed queue*
   - *checkPalindrome() # Checks if the Queue is a palindrome or not*

   Write a class TestApp to check the functionalities of **QueueReverse** and **StackReverse** and print if the Stack or Queue is a palindrome or not.
   Note: The input to your program is N natural numbers or characters.
   *(*throws *NotANumberException, NotAStringException* and *NegativeNumberException* based on the input*)*

   Input Format
   First line is **S** (Stack Operation) or **Q** (Queue Operation) followed by the number of elements **N.** (throw INVALID for invalid input)
   Second line has **N** elements.

   Output Format
   First line prints **STACK** if input is **S** or **QUEUE** if input is **Q**
   Second line prints the input elements
   Based on the input, third line prints the reversed elements of the stack or queue
   Prints **Yes** if the reversed stack/queue is palindrome and **No** if not a palindrome.

   TEST CASES

INPUT	OUTPUT
S 5 M A c A M	STACK M A c A M M A c A M YES

Q 3 1 -1 1	QUEUE NegativeNumberException

2. Create an Interface **BillGenerator** with the functions *generateBill()* and *displayBill()*.
Create two classes **ElectricityBill** and **PhoneBill** that implements the **BillGenerator**.

Class **ElectricityBill** consists of the following attributes:

- *consumerNumber* #if invalid throw *NotANumberException*
- *consumerName* #if invalid throw *NotAStringException*
- *previousMonthReading* #if invalid throw *NotANumberException*
- *currentMonthReading* #if invalid throw *NotANumberException*
- *typeOfConnection #(i.e., domestic or commercial) if invalid* throw *NotAValidConnection*

Compute the electricity bill amount using *generateBill()* method in **ElectricityBill,** using the following formula:
Total Number of units = *currentMonthReading - previousMonthReading*

Bill amount for a Connection = (No.of units in slab 1 *Rs per unit) + (No.of units in slab 2 * Rs per unit) + (No.of units in slab 3 * Rs per unit) + (No.of units in slab 4 * Rs per unit)

The slabs for Domestic connection and Commercial Connection is given in the following table:

Slab no	DOMESTIC CONNECTION		COMMERCIAL CONNECTION	
	Slab	**Rs. per Unit**	**Slab**	**Rs. per Unit**
Slab 1	0-50 Units	Rs 3.00	0-300 Units	Rs 5.50
Slab 2	51-100 Units	Rs 3.15	301-400 Units	Rs 6.60
Slab 3	101-150 Units	Rs 3.80	401 -500 Units	Rs 7.70
Slab 4	Above 151 Units	Rs 4.00	Above 501 Units	Rs 8.00

Example:
For Domestic Connection
If the total unit used is 100
*Bill amount is calculated as: 50*3.00 + 50*3.15= 307.50*
If the total unit used is 50
*Bill amount is calculated as: 50*3.00 = 150.00*

For Commercial Connection
If the total unit used is 300
*bill amount is calculated as: 300*5.50 =1650.00*
If the total unit used is 453
*Bill amount is calculated as: 300*5.50+100*6.60+53*7.70 =2718.10*

Use the *displayBill()* method in **ElectricityBill** to display the Electricity Bill Details.

Class **PhoneBill** consists of the following attributes:

- *customerId* #if invalid throw *NotANumberException*
- *customerName* #if invalid throw *NotAStringException*
- *totalMinutes* #if invalid throw *NotANumberException*

Compute the phone bill using *generateBill()* method in **PhoneBill** using the following formula:

Bill generated = FixedCharge + Call rate*number of minutes (*# Fixed Charge is 250*)
The Call Rate is given as:
First 100 minutes - Rs 0
After 100 minutes - Rs 1.50 per Minute

displayBill() is a member function in **PhoneBill** which is used to display the Phone Bill Details.

Create a Class **TestBill** to test the functionality of the **ElectricityBill** and the **PhoneBill** Classes

Input Format

The first line consists of either **E** followed by a space **<C or D >** or **P,** where **E** is Electricity Bill and **C** stands for Commercial and **D** for Domestic. **P** stands for Phone Bill. If invalid input, print INVALID.

If Input type is **E <C or D>**, then
Second line is *consumerNumber.* if invalid throws *NotANumberException.*
Third line is *consumerName.* if invalid throws *NotAStringException.*
Fourth line is *previousMonthReading.* if invalid throw *NotANumberException.*
Fifth line is *currentMonthReading.* if invalid throw *NotANumberException.*

If Input type is **P**, then
Second line is *customerId.* if invalid throw *NotANumberException.*
Third line is *customerName.* if invalid throw *NotAStringException.*
Fourth line is *totalMinutes.*

Output Format
Based on input, the output is as follows:
First line contains Consumer ID/ Customer ID
Second line contains Consumer Name/Customer Name
Number of Units or Total Minutes **to be charged**
Total Bill Amount (Rounded to two decimal places)

TEST CASES

INPUT	OUTPUT
E D 123 Ram 100 150	123 Ram 50 150.00
P 809 Alok 500	809 Alok 400 850.00
E C 1234 200	NotAStringException

3. The phone book contains the contact details of different persons. Each contact detail in the phone book keeps track of the name of the person, and associates the name with one phone number(A ten digit number). Identify the relationship between phone book and contact details, and properly organize the attributes.
   The **PhoneBook** has the following functionalities:

   - ***PhoneBook() :-*** *creates a new empty phone book.*
   - ***insertEntry():*** *Adds a new entry at the end of the phone book, if there is no contact already with the new contact's name and phone number* .
   - ***lookUp(substring) :*** *Searches the phon*e book for the contacts with the name containing the given substring.

   Input Format

   - First line contains the number of phone book entries, **N**.
   - Next N lines of the input are space separated values ***<String> <Integer>***
   - After that there can be multiple lines which look up for contact information.
     Each line inputs **P** <**substring>,** where P indicates lookup.

   Output Format

   - If the input line contains **P** <**substring>,** then
     - print all the contacts whose name starts with <substring>. Each contact should be displayed in each line in the format <name> <phone-number>
     - print **NoSuchEntry** - If no contact is found in the phone book whose name starts with the given substring.

   Exceptions:-

   - Print **IncorrectPhonenumber** and terminate program **-** If the phone number is invalid.

TEST CASES

INPUT	**OUTPUT**
6 Mukesh Kumar 9717635807 Anil Sharma 9810747515 Brijesh Kumar 9811296965 Sunita Sharma 8800767646 Kumar Karthik 8956774210 Dhiraj Kr. Sharma 9868585688 P Kumar P Raj	Kumar Karthik 8956774210 NoSuchEntry