1. JavaScript: Notes Store

In this challenge, the task is to create a class NotesStore. The class will manage a collection of notes, with each note having a state and a name. Valid states for notes are: 'completed', 'active', and 'others'. All other states are invalid.

The class must have following methods:

1. addNote(state, name): adds a note with the given name and state to the collection. In addition to that:
   - If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
   - If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.
2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▼ Input Format For Custom Testing

In the first line, there is an integer, *n*, denoting the number of operations to be performed.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains space-separated strings such that the first of them is a function name, and the remaining ones, if any, are parameters for that function.

▼ Sample Case 0

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Explanation
For all 3 *addNote* operations, the *addNote* function is called with a state and a name. Then, the *getNotes* function is called for 'active' state and 'completed' state respectively, and the result is printed. Then, the *getNotes* function is called for 'foo' state, which throws an error since this state is invalid, and the error is printed.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
addNote active
addNote foo Study
getNotes completed
```

Sample Output

```
Error: Name cannot be empty
Error: Invalid state foo
No Notes
```

Explanation
The *addNote* function is called with 'active' state and an empty name, which throws an error since the name is empty. Then, *addNote* is called with 'foo' state, which throws an error since the state is invalid and an error is thrown. Finally, the *getNotes* function is called for 'completed' state, an empty array is returned since no note exists in this state, and 'No Notes' is printed by the stubbed code.