OOP Advanced Sample Exam – Lambda Core

You have been given a partition of the code architecture of the software that controls the functionality of Lambda Core – a super, high-tech, nuclear power plant. The code you have been given, however, is broken and poorly written. You will have to refactor and fix it before you can begin your main work, which is completing the code so that it could define the full functionality of Lambda Core.

Overview

Lambda Core's functionality is not that complex. Lambda Core supports the addition and removal of cores, which drive the main logic of the power plant. The cores hold fragments – each fragment has its own role in the power plant logic.

When adding cores, each new core has its own unique name, which is a **capital letter** from the English Alphabet. The first core is given the letter **"A"** and every core created **after that**, is given **the letter following** that of **the one before** it.

A core has a **type**, **durability**, and several **fragments**. Each fragment **affects the pressure** on the core, somehow. There are fragments which **increase** the pressure and there are such which **decrease** it. When the pressure is **above 0** it starts **decreasing** the **core's durability**. When that happens, the core enters **critical state**. When fragments are detached, the **pressure on the core lowers**, which means that the core's durability **increases** again. Note that the durability of the cores should **never fall below zero**.

Fragments have a **name, type** and **pressure affection**. The pressure affection depends on the type of the fragment.

Models

There are mainly two types of Cores and two types of Fragments. The Cores are:

- **SystemCore**
  - A basic core with **durability,** equal to the given from the input.
- **ParaCore**
  - A core with **durability**, equal to the given from the input, **divided by 3**.

The two types of Fragments are:

- **NuclearFragment**
  - A fragment with **pressure affection**, equal to the given from the input, **multiplied by 2**.
  - **Increases** the pressure **on the core** by the **value** of its **pressure affection**.
- **CoolingFragment**
  - A fragment with **pressure affection**, equal to the given from the input, **multiplied by 3**.
  - **Decreases** the pressure **on the core** by the **value** of its **pressure affection**.

Commands

- **CreateCore:@type@durability**
  - Creates a core with the **given type** and **given durability**, and adds it to the power plant.

    - The **type** needs to be either **"System"**or **"Para"**, otherwise the command should fail.
    - The **GIVEN durability** needs to be **positive** (or zero), otherwise the command should fail.
    - In case of success, the command should print a message **"Successfully created Core {coreName}!".**
    - In case of failure, the command should print a message **"Failed to create Core!".**
- **RemoveCore:@name**
    - Removes the core with the **given name** from the power plant.
    - If there is **no such** core, with the **given name**, the command should fail.
    - In case of success, the command should print a message **"Successfully removed Core {coreName}!".**
    - In case of failure, the command should print a message **"Failed to remove Core {coreName}!".**
- **SelectCore:@name**
    - Selects the core with the **given name**, and **all fragment attachment commands** will be performed on the **currently selected core**.
    - If there is **no such** core, with the **given name**, the command should fail.
    - In case of success, the command should print a message **"Currently selected Core {coreName}!".**
    - In case of failure, the command should print a message **"Failed to select Core {coreName}!".**
- **AttachFragment:@type@name@pressureAffection**
    - Creates a fragment with the **given name** and **pressure affections** and attaches it to the **currently selected core**.
    - The **type** needs to be either **"Nuclear"**or **"Cooling"**, otherwise the command should fail.
    - The **GIVEN pressure affection** needs to be **positive** (or zero), otherwise the command should fail.
    - If there is **NO currently selected core**, the command should fail.
    - In case of success, the command should print a message **"Successfully attached Fragment {coreName} to Core {currentlySelectedCore}!".**
    - In case of failure, the command should print a message **"Failed to attach Fragment {fragmentName}!".**
- **DetachFragment:**
    - Detaches the last fragment from the **currently selected core**.
    - If there is **no such** fragment, with the **given name**, the command should fail.
    - If there is **NO currently selected core**, the command should fail.
    - In case of success, the command should print a message **"Successfully detached Fragment {fragmentName} from Core {coreName}!".**
    - In case of failure, the command should print a message **"Failed to detach Fragment {fragmentName}!".**
- **Status:**
    - Prints status information about the Power plant.
    - The information should be in the following format:
    - **"Lambda Core Power Plant Status:**
    - **Total Durability: {totalDurability}**
    - **Total Cores: {countOfCores}**
    - **Total Fragments: {countOfAllFragments}**
    - **Core {coreName}:**
    - **####Durability: {currentDurability}**
    - **####Status:{NORMAL / CRITICAL}**
    - **Core {coreName}:"**
    - …

The input sequence ends when the command **"System Shutdown!"** is entered. The parameters are **not annotations**.

Input

- The input will consist only of the commands specified above.
- The input ends when the command **"System Shutdown!"** is entered.

Output

- As output you must print only the messages that are specified in the Commands section above.

Constraints

- The durability of the cores and the pressure affection of the fragments are integers in range [0, 2^{31}- 1].
- There will be a **maximum** of **26** cores, **created** from the **input**.
- The names of the fragments will be strings.
- The input will always be in the format specified above.
- Allowed time/memory: 100ms/16MB.

Examples

Input	Output
`CreateCore:@System@2000`	`Successfully created Core A!`
`CreateCore:@System@1000`	`Successfully created Core B!`
`AttachFragment:@Cooling@A32@100`	`Failed to attach Fragment A32!`
`SelectCore:@A`	`Currently selected Core A!`
`AttachFragment:@Cooling@A64@200`	`Successfully attached Fragment A64`
`AttachFragment:@Cooling@A72@300`	`to Core A!`
`Status:`	`Successfully attached Fragment A72`
`SelectCore:@B`	`to Core A!`
`AttachFragment:@Nuclear@B12@250`	`Lambda Core Power Plant Status`
`Status:`	`Total Durability: 3000`
`System Shutdown!`	`Total Cores: 2`
	`Total Fragments: 2`
	`Core A:`
	`####Durability: 2000`
	`####Status: NORMAL`
	`Core B:`
	`####Durability: 1000`
	`####Status: NORMAL`
	`Currently selected Core B!`
	`Successfully attached Fragment B12`
	`to Core B!`
	`Lambda Core Power Plant Status`
	`Total Durability: 2500`
	`Total Cores: 2`
	`Total Fragments: 3`
	`Core A:`
	`####Durability: 2000`
	`####Status: NORMAL`
	`Core B:`
	`####Durability: 500`
	`####Status: CRITICAL`

Use of given code

Use the given **LStack** for the main functionality of fragments in the cores.

Refactoring

Refactor the code you have received, so that it follows the best practices, you've learned in this course. Also, refactor your own code, after writing it, so that you remove any mistakes you've made.

Bug Fixing

Fix all the bugs with the given code, and make it so that it would work with the business case.

Unit Testing

Write unit tests for all the methods of the **LStack** class.

Write unit tests for the functionality of the **Select Command**.

Write unit tests for the **AttachFragment** and **DetachFragment Commands**.

Reflection

Use reflection at least once, in your code, and make it so that it is essential to the main logic.