**OOP Retake Exam – Avatar**

The nations are at an eternal conflict. Which one of them has the strongest benders? Which one of them has the strongest deities. Who will win and who will fail, trying to do so. . . You decide!

**Overview**

You have been given the godlike power to supervise the conflict between the nations. However, you are lazy to do it by hand, so you will write a godlike program to do it.

**Structure**

The structure of the program consists of Benders and Monuments. Implement your classes, in the same way as they are **described** **below** (class names, field types, field names etc.). Keep the **naming conventions** and **rules**.

**Bender**

All Benders have 2 characteristics you must implement:

* **Name** – a **string**, holding the name of the Bender.
* **Power** – an **integer**, holding the power of the Bender.

There are **4 types** of Benders:

**AirBender**

The **AirBender** has an additional characteristic:

* **AerialIntegrity** – a **floating-point number**, holding the **aerialIntegrity** of the Bender.

**WaterBender**

The **WaterBender** has an additional characteristic:

* **WaterClarity** – a **floating-point number**, holding the **waterClarity** of the Bender.

**FireBender**

The **FireBender** has an additional characteristic:

* **HeatAggression** – a **floating-point number**, holding the **heatAggression** of the Bender.

**EarthBender**

The **EarthBender** has an additional characteristic:

* **GroundSaturation** – a **floating-point number**, holding the **groundSaturation** of the Bender.

**Monuments**

The second entities are the **Monuments**. A monument has 1 characteristic:

* **Name** – a **string**, holding the name of the Monument.

There are **4 types** of Monuments:

**AirMonument**

The **AirMonument** has an additional characteristic:

* **AirAffinity** – an **integer**, holding the **airAffinity** of the **Monument**.

**WaterMonument**

The **WaterMonument** has an additional characteristic:

* **WaterAffinity** – an **integer**, holding the **waterAffinity** of the **Monument**.

**FireMonument**

The **FireMonument** has an additional characteristic:

* **FireAffinity** – an **integer**, holding the **fireAffinity** of the **Monument**.

**EarthMonument**

The **EarthMonument** has an additional characteristic:

* **EarthAffinity** – an **integer**, holding the **earthAffinity** of the **Monument**.

**Nation**

The **nations** are just the **majorities** of **Benders**. For example, the **Fire nation** **represents** a **collection** of all the created **Fire Benders**. You do **NOT** need a class for the nations, although it won’t be incorrect if you implement a such.

**Functionality**

The functionality and main logic of the program circles around the benders and the monuments. The benders don’t just stand there and do nothing. The four types of benders represent the warriors of the different nations. Each nation can issue a war and involve all the others in a fight, which results in only the strongest nation – remaining alive. The monuments also have a role in these wars. Upon starting a war, all monuments are activated, which by itself increases the nations’ total power. You can see below, in the subsections, exactly how, this happens.

**Classes**

**The Controller Class**

The business logic of the program should be concentrated around several commands. Implement a class called **NationsBuilder**, which will hold the **main functionality**, represented by these **public** **methods**:

|  |
| --- |
| **NationsBuilder.cs** |
| void AssignBender(List<string> benderArgs)  {  //TODO: Add some logic here …  }  void AssignMonument(List<string> monumentArgs)  {  //TODO: Add some logic here …  }  string GetStatus(string nationsType)  {  //TODO: Add some logic here …  }  void IssueWar(string nationsType)  {  //TODO: Add some logic here …  }  string GetWarsRecord()  {  //TODO: Add some logic here …  } |

**Benders**

The Benders’ main role is to be the source of power for a certain nation. They are the warriors that define the power of their nation. The total power of a nation is used when a war is issued. The nations’ total power is compared and the strongest nation survives.

A Bender’s total power is calculated by the following formulas:

* **AirBender** – **(power \* aerialIntegrity)**
* **WaterBender** – **(power \* waterClarity)**
* **FireBender** – **(power \* heatAggression)**
* **EarthBender** – **(power \* groundSaturation)**

A **nation’s total power** represents the **sum** of its **Benders’ total powers**.

**Monuments**

The monuments increase a nation’s total power, when a war is issued. This means that you must **first** **calculate** the **total power** of **each nation**, and then **increase** it by the **bonuses** of the **monuments**.

Every monument has an **affinity**. The **sum** of **ALL** monuments’affinities **increases** the nation’s **total power** by **percentage**.

Each monument **corresponds** to a **particular nation**. If you have an **AirMonument** it affects the **Air nation** **ONLY**.

For example: If the Air nation has **1000 total power** and you have 2 **AirMonuments** – one with **100 airAffinity** and the other with **200 airAffinity**. The **sum** of their affinities is **300 airAffinity**. That means you must **increase** the **total power** of the **Air nation** by **300%**.

So you must **increase** the **total power** by **(1000 / 100) \* 300** = **3000**. That would result in **1000** + **3000** = **4000**.

**Commands**

There are several commands that must be implemented in order for the program to follow the business logic.

You will see their functionality and what they are supposed to do in this section.

**Bender Commands**

Parameters – **type** (**string**), **name** (**string**), **power** (**int**), **secondaryParameter** (**floating-point number**).

Creates a **Bender** of the **given type**, with the **given parameters**.

The **type** will either be “**Air**”, “**Water**”, “**Fire**” or “**Earth**”.

The **secondaryParameter** should be assigned as a value, **depending** of the **type** of the **Bender**, to one of the following:

* **aerialIntegrity**
* **waterClarity**
* **heatAggression**
* **groundSaturation**

**Monument Command**

Parameters – **type** (**string**), **name** (**string**), **affinity** (**int**).

Creates a **Monument** of the **given type**, with the **given parameters**.

The **type** will either be “**Air**”, “**Water**”, “**Fire**” or “**Earth**”.

**Status Command**

Parameters – **nation** (**string**).

The **nation** will either be “**Air**”, “**Water**”, “**Fire**” or “**Earth**”.

Prints detailed information about the **nation**, corresponding to the **given name**. Prints a string representation of all the **Benders** and all of the **Monuments** corresponding to the **given nation**.

**War Command**

Parameters – **nation** (**string**).

The **nation** will either be “**Air**”, “**Water**”, “**Fire**” or “**Earth**”.

The **given nation**, issues a war and **involves all other nations** in it. All nations’ **total** **power** is calculated, **summing** up the **total power** of their **Benders**, and then adding up **all bonuses** from their **Monuments**.

When everything has been calculated, the nation with the **highest** **total** **power** is the one that **wins**. The **other** **nations** must **delete** **all** of their **Benders** and **Monuments**, as if they were destroyed.

Naturally, the nations that lost, **may** still **add new Benders** and **Monuments**, so that the logic of the program may continue.

You can safely assume, that there will be **NO nations** who share the **SAME total power** as **value**.

**Quit Command**

Terminates the user input sequence and **exits** the program. Upon doing so, you should print **every issued war** and the **nation** that issued it. The wars must be printed by order of issuing, or in other words, by **order of input**.

**Input**

The input consists of several commands, as you’ve already seen. Their input format is described below:

* **Bender {type} {name} {power} {secondaryParameter}**
* **Monument {type} {name} {affinity}**
* **Status {nation}**
* **War {nation}**
* **Quit**

**Output**

This is the output section. This section describes the format of the output you must present. You can see how the classes are being represented as strings and what is the format of the output of the commands.

**Classes**

Each **type** of **Bender** has his own **string representation**:

* **Air Bender: {benderName}, Power: {power}, Aerial Integrity: {aerialIntegrity}**
* **Water Bender: {benderName}, Power: {power}, Water Clarity: {waterClarity}**
* **Fire Bender: {benderName}, Power: {power}, Heat Aggression: {heatAggression}**
* **Earth Bender: {benderName}, Power: {power}, Ground Saturation: {groundSaturation}**

Each **type** of **Monument** has his own **string representation**:

* **Air Monument: {monumentName}, Air Affinity: {airAffinity}**
* **Water Monument: {monumentName}, Water Affinity: {waterAffinity}**
* **Fire Monument: {monumentName}, Fire Affinity: {fireAffinity}**
* **Earth Monument: {monumentName}, Earth Affinity: {earthAffinity}**

**Commands**

**Status Command**

Prints all of its Benders in the following way:

**{nationType} Nation  
Benders:  
###{bender1}  
###{bender2}  
. . .  
Monuments:  
###{monument1}  
###{monument2}  
. . .**

The **nation type** will either be “**Air**”, “**Water**”, “**Fire**” or “**Earth**”, depending on the nation that is being presented.

Each **Bender** and **Monument** must be **printed** with its **string representation**.

The data should **NOT** be ordered (in other words – **order of input**).

In case there are **NO Benders** or **NO Monuments** the output should look like this:  
**Benders: None**  
or  
**Monuments: None**

**Quit Command**

Terminates the program and prints **ALL** the **issued wars**, **with the nations** that **issued** them, in the following format:

**War 1 issued by {nation}.  
War 2 issued by {nation}.  
War 3 issued by {nation}.  
. . .  
War N issued by {nation}.**

The order of printing is by **order** of **issuing** the wars – from the **first issued**, to the **last one**.

**NOTE**: **All floating-point numbers** in the **output** should be **FORMATTED** to the **second digit** after the **decimal point**.

**Constrains**

* The **names** of the **Benders** and the **Monuments** will contain only **alphanumeric characters**.
* All **integers** in the input will be in **range [0, 1.000.000.000]**.
* All **floating-point numbers** in the input will be in **range [0, 1.000.000]**.
* There will be **NO invalid** input lines.
* There will be **NO invalid parameters** in the commands.
* There will be **NO stalemates** when issuing a war.
* There will **ALWAYS** be **atleast 1** war.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bender Fire Tony 1000 2500.5667  Bender Fire Donald 1000 2100  Bender Air Yu 1000 1230  War Fire  War Air  War Fire  Quit | War 1 issued by Fire  War 2 issued by Air  War 3 issued by Fire |
| Bender Air Yu 100 215.677  Bender Air Muk 200 241.24124  Bender Fire Donald 100 214.4  Monument Fire JerseyGrew 1000  Status Air  Status Fire  War Fire  Status Air  Bender Earth Dock 201 123  Bender Water Klci 201 1244  War Water  Quit | Air Nation  Benders:  ###Air Bender: Yu, Power: 100, Aerial Integrity: 215.68  ###Air Bender: Muk, Power: 200, Aerial Integrity: 241.24  Monuments: None  Fire Nation  Benders:  ###Fire Bender: Donald, Power: 100, Heat Aggression: 214.40  Monuments:  ###Fire Monument: JerseyGrew, Fire Affinity: 1000  Air Nation  Benders: None  Monuments: None  War 1 issued by Fire  War 2 issued by Water |

**Tasks**

These are the tasks you must complete in order to solve the exam completely. Submit your solutions in the Judge system. **NOTE**: Every task has its **own execution strategy**, so check them out.

**Structure**

**Implement** the described **classes**, in the way they were described (**class names**, **field names** and so on). Create a good OOP structure with packages and distribution of classes.

Put everything in a global package. . . Example: “**src/avatar/. . .**”, and submit the **PACKAGE ONLY** (“**avatar**”) it in the Judge System. Do **NOT** submit the whole “**src**” folder.

**Input / Output (I/O)**

Implement the **functionality** that was **described** (**Commands**, **class inner functionality** etc.). Check with the **Example tests**, if you’ve done it **correctly**.

You have a few tests in Judge, which mainly **test** the **correct functionality**.

For **this task**, submit the **whole** “**src**” folder, in the Judge System.