**OOP Basics Exam – Minedraft**

You ever heard about the Rick and Morty’s Foundation for mining Plumbus Ore. Naaa, probably not. Well let’s just say there is this company that mines things, and they hired you to write them a supervising software. A draft which will be used to analyze the data of the mining – a … **Minedraft**.

**Overview**

The main system consists of Harvesters and Providers. The Harvesters are the ones that make real money – they mine Plumbus Ore. But they need a large amount of energy to do that. That’s where the Providers come. The Providers are the ones that provide the energy for the harvesters.

**Task 1: Structure**

The Structure consists of Harvesters and Providers.

**Harvesters**

A basic **Harvester** has the following properties:

* **id** – a **string**.
* **oreOutput** – a **floating-point number**.
* **energyRequirement** – a **floating-point number**.

For all harvesters you **need to validate**, that **ore output** and **energy requirement** for each harvester **is NOT negative**. Also you need to **validate** that **energy requirement** for each harveter **is NOT over 20000**. There are generally **2** types of **Harversters**:

**SonicHarvester**

Really fast… Has an extra property:

* **sonicFactor** – an **integer**.

**UPON INITIALIZATION**, **divides** its given **energyRequirement** by its **sonicFactor**.

**HammerHarvester**

Heavy... and big.

**UPON INITIALIZATION**, **increases** its **oreOutput** by **200 %,** and **increases** its **energyRequirement** by **100 %**.

**Providers**

A basic **Provider** has the following properties:

* **id** – a **string**.
* **energyOutput** – a **floating-point number**.

Every provider energy output **need to be positive numbers,** **less than 10000**. There are generally **2** types of **Providers**:

**SolarProvider**

Extracts energy from the Sun. Nothing special here.

**PressureProvider**

Extracts energy from deep beneath the earth. Temperatures and Pressure affect it.

**UPON INITIALIZATION**,increases its **energyOutput** by **50 %**.

**Task 2: Business Logic**

**The Controller Class**

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

|  |
| --- |
| **DraftManager.cs** |
| string RegisterHarvester(List<string> arguments)  {  //TODO: Add some logic here …  }  string RegisterProvider(List<string> arguments)  {  //TODO: Add some logic here …  }  string Day()  {  //TODO: Add some logic here …  }  string Mode(List<string> arguments)  {  //TODO: Add some logic here …  }  string Check(List<string> arguments)  {  //TODO: Add some logic here …  }  string ShutDown()  {  //TODO: Add some logic here …  } |

**NOTE: DraftManager class** methods are called from the outside so these methods **must** **NOT** receive the command parameter as part of the arguments!

**Functionality**

The whole system works on **3 modes** – “**Full Mode**”, “**Half Mode**”, “**Energy Mode**”. Depending on the mode, the **Harvesters** and **Providers** work differently. By **DEFAULT** the mode is “**Full Mode**”.

The **Providers** and **Harvesters** have **ids**, which will **always** be **unique**.

When a **day passes**, the **Providers** produce energy and the **Harvesters** consume energy and mine Plumbus Ore. In your program a **day passes** when you have been given the **corresponding command**.

The **Providers** produce energy which is being stored on the system. When there is **ENOUGH** **energy** to **power up** **ALL** **Harvesters**, the **Harvesters** **consume** it and return the ore.

The system keeps the **totalStoredEnergy** and the **totalMinedOre**.

**Modes**

The different modes make the **Harvesters** work differently. You can save more power by changing the modes and stalling them a little. The **Providers** remain **unaffected** by the modes.

**Full Mode**

All **Harvesters** consume their **FULL energy requirements**, and produce their **FULL ore output**.

**Half Mode**

All **Harvesters** consume **60 %** of their **energy requirements**, and produce **50 %** of their **ore output**.

**Energy Mode**

The **Harvesters** consume **nothing**, and produce **nothing**. They practically do **NOT** work.

**Commands**

There are several commands that control the business logic of the application you are supposed to build.   
They are stated below.

**RegisterHarvester Command**

Creates a **Harvester**, and registers it into the system, so they can start mining when new day come.

**Parameters**

* **type** – a **string**, equal to either **Sonic** or **Hammer**.
* **id** – a **string**.
* **oreOutput** – a positive **floating-point number**.
* **energyRequirement** – a positive **floating-point number**.

If the type is **Sonic**, you will receive **1 extra parameter**:

* **sonicFactor** – a positive **integer**.

**RegisterProvider Command**

Creates a **Provider**, and registers it into the system. They start to provide energy from next day.

**Parameters**

* **type** – a **string**, equal to either **Solar** or **Pressure**.
* **id** – a **string**.
* **energyOutput** – a positive **floating-point number**.

**Day Command**

When you receive this command a day passes. This is the moment where real work starts. You need to **calculate** all the provided **energy** and **STORE** it in the system. Then you need to **check** if there is **enough** **energy** for harvesters to start mining. If the sum of energy requirement of **ALL** harvesters is more than the **energy** **stored** then **NOTHING** happens. If there is **enough energy**, **ALL** harvesters **start mining** and they **consume** from the **stored** **energy** **EQUAL** to their energy requirement.

**NOTE**: The summed up **energyRequirement** might be **less** or **more** depending on the current **working** **Mode**.

**Mode Command**

Changes the **mode** of the system, to the **given one**.

**Parameters**

* **mode** - a **string**, equal to either **Full**, **Half** or **Energy**.

**Check Command**

**Checks** the **Provider** or the **Harvester** with the **given id**, returning a **string representation** of it. The system should check if there is an **element** with the **given id** among the **Providers** or the **Harvesters**. The **ids** are **unique** so there should be only **one** with that **id**.

**Parameters**

* **id** – a **string**.

**Shutdown Command**

**Ends** the program and **print total energy stored and ore mined**

**Task 3: Input / Output**

**Input**

Below, you can see the **format** in which **each command** will be given in the input:

* **RegisterHarvester {type} {id} {oreOutput} {energyRequirement}**
* **RegisterHarvester Sonic {id} {oreOutput} {energyRequirement} {sonicFactor}**
* **RegisterProvider {type} {id} {energyOutput}**
* **Day**
* **Mode {mode}**
* **Check {id}**
* **Shutdown**

**Output**

Below you can see what output should be provided from the commands.

**RegisterHarvester Command**

Successful command should print “**Successfully registered {type} Harvester – {id}**”.

Unsuccessfull comand: "**Harvester is not registered, because of it's {propertyName}**"

**RegisterProvider Command**

Should output a message “**Successfully registered {type} Provider – {id}**”.

Unsuccessfull comand: "**Provider is not registered, because of it's {propertyName}**"

**Day Command**

Should output a message

“**A day has passed.**

**“Energy Provided: {summedEnergyOutput}**”.

“**Plumbus Ore Mined: {summedOreOutput}**”.

The **summedEnergyOutput** and **summedOreOutput** are the ones that have been mined for the day.

**Mode Command**

Should output a message “**Successfully changed working mode to {mode} Mode**”.

**Check Command**

Should return a **string representation** of the element with the **given id**. If there is no such element, the command should output a message “**No element found with id – {id}**”.

Because the element can either be a **Provider** or a **Harvester**, both **output formats** have been provided below:

|  |  |
| --- | --- |
| **Harvester** | **Provider** |
| **“{type} Harvester – {id}**  **Ore Output: {oreOutput}**  **Energy Requirement: {energyRequired}”** | **“{type} Provider – {id}**  **Energy Output: {energyOutput}”** |

**Shutdown Command**

Should output a message

“**System Shutdown**

**Total Energy Stored: {totalEnergyStored}**

**Total Mined Plumbus Ore: {totalMinedOre}**”.

The **totalEnergyStored** and **totalMinedOre** are the total values that have been gathered throughout the program’s execution.

**Constraints**

* The **id** will be a string which may contain any ASCII character, except **space** (‘ ’).
* The **ids** will always be **unique**.
* All **floating-point numbers** will be in **range [-1.000.000, 1.000.000]**.
* The **sonicFactor** will be in **range [1, 10]**.
* There will be **NO invalid** input data.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| RegisterHarvester Sonic AS-51 100 100 10  RegisterHarvester Hammer CDD 100 50  RegisterProvider Solar Falcon 100  Day  Check AS-51  Check CDD  Check Falcon  Day  Shutdown | Successfully registered Sonic Harvester - AS-51  Successfully registered Hammer Harvester - CDD  Successfully registered Solar Provider - Falcon  A day has passed.  Energy Provided: 100  Plumbus Ore Mined: 0  Sonic Harvester - AS-51  Ore Output: 100  Energy Requirement: 10  Hammer Harvester - CDD  Ore Output: 300  Energy Requirement: 100  Solar Provider - Falcon  Energy Output: 100  A day has passed.  Energy Provided: 100  Plumbus Ore Mined: 400  System Shutdown  Total Energy Stored: 90  Total Mined Plumbus Ore: 400 |
| RegisterHarvester Sonic AS-51 100 1000000 10  RegisterHarvester Hammer CDD 100 50  RegisterProvider Solar Falcon 100  RegisterProvider Solar Pesho 100000  Day  Check CDD  Check Falcon  Day  Shutdown | Harvester is not registered, because of it's EnergyRequirement  Successfully registered Hammer Harvester - CDD  Successfully registered Solar Provider - Falcon  Provider is not registered, because of it's EnergyOutput  A day has passed.  Energy Provided: 100  Plumbus Ore Mined: 300  Hammer Harvester - CDD  Ore Output: 300  Energy Requirement: 100  Solar Provider - Falcon  Energy Output: 100  A day has passed.  Energy Provided: 100  Plumbus Ore Mined: 300  System Shutdown  Total Energy Stored: 0  Total Mined Plumbus Ore: 600 |
| RegisterProvider Pressure Deep-1 1000  RegisterProvider Pressure Deep-3 2000  Day  Mode Energy  RegisterHarvester Hammer S-1 10000 11250  Day  Check Something  Check S-1  Mode Half  Day  Shutdown | Successfully registered Pressure Provider - Deep-1  Successfully registered Pressure Provider - Deep-3  A day has passed.  Energy Provided: 4500  Plumbus Ore Mined: 0  Successfully changed working mode to Energy Mode  Harvester is not registered, because of it's EnergyRequirement  A day has passed.  Energy Provided: 4500  Plumbus Ore Mined: 0  No element found with id - Something  No element found with id - S-1  Successfully changed working mode to Half Mode  A day has passed.  Energy Provided: 4500  Plumbus Ore Mined: 0  System Shutdown  Total Energy Stored: 13500  Total Mined Plumbus Ore: 0 |

**Task 4: Bonus**

**Abstraction**

Probably, you have already noticed, that there is a way to **improve** the **abstraction** of your code. If NOT, now is the time to think about this. For this task, you need **one more level of abstraction for Harvester and Providers**.

**Factories**

You know, that the keyword **new** is a bottleneck and we are trying to use it as less as possible. We even try to separate it in new classes. These classes are called Factories and the convention for them is **{TypeOfObject}Factory.** You need to have **two** **different** **factories**, **one for Harvesters and one for Providers**. This is actually a design pattern and you can read more about it. [Factory Pattern](https://www.dotnetperls.com/factory)

**Points**

For all tasks you can submit same project. Every different task give you points:

* 100 points
* 200 points
* 100 points
* 50 points