Object-Oriented Programming – Practical Exam

Problem 2. Infestation

The Zemyans ("Zemya" coming from the Bulgarian word, meaning the same as "Terra" in Latin) have captured some renegade Grez. The Grez are a slimy alien race, which increase their ranks through a process called infestation. Now the Zemyans want to simulate the Grez interactions with their own units, to have a better fighting chance when the invasion comes.

You are given an API, which supports some basic unit interactions, based on early Zemyan designs, before the Grez showed up. You need to extend the API to match the current situation.

There are some simple rules the API supports:

- There are three main components of the API: the **HoldingPen**, the **Unit** and the **Supplement**
  - A **HoldingPen** contains units in an isolated environment and executes basic interactions between them. It can also add new units, provide them with supplements and report the their status
  - A **Unit** represents any active thing in the game – e.g. a **Person**, a **Dog**, a **Tank**, etc.
    - Units can interact with each other and their interactions depend on several characteristics (discussed later)
  - A Supplement is an object, which quantitatively changes the basic characteristics of an object. E.g. a **Weapon** is a kind of a **Supplement**, which makes a trained user more dangerous
- **Units**
  - Every unit has an **Id** (a string name) which uniquely identifies it
  - Every unit has base **Health**, **Power** and **Aggressiveness**
    - Base meaning the value before adding supplements, which can increase the aforementioned
  - The more health a unit has, the more attacks it can survive
  - The more power a unit has, the more potent its attacks are
  - The more aggressive a unit is, the more likely it is to attack another unit
  - Every unit has a set of active supplements (usually empty when first created)
  - Every unit has a classification (type) either **Biological**, **Mechanical** or **Psionic** (i.e. telepathic)
- **Supplements**
  - Every **Supplement** can give a bonus (positive or negative) to a units **Health**, **Power** and **Aggressiveness**
  - The API currently has NO implemented supplements, but has infrastructure which can be extended to include supplements
- The **HoldingPen** is the place where all operations (commands from the console) are parsed and executed

Commands

There are four commands the **HoldingPen** supports:

- **Insertion** command – creates Units
  - Syntax: `insert Dog dogName`

- Inserting (creating) a Unit requires its type and its id
- Syntax: **`insert Dog sharo`** – creates a Dog with the id "sharo"

- **Proceed** command – forces all units to engage in interactions
  - Syntax: **`proceed`**
  - During a proceed command, each unit receives information about all the other units in the HoldingPen
  - During a proceed command, each unit can interact exactly once
    - The unit gives information to the HoldingPen about the interaction it wishes to execute and the HoldingPen takes care of the actual interaction
  - Interactions are three types – **attack**, **infest** and the **passive interaction** (i.e. the unit does nothing)
- **Supplement** command – adds a supplement to an existing unit
  - Syntax: **`supplement SupplementType targetUnitId`**
  - Creates a supplement of the desired type (e.g. AggressionInhibitor) and adds it to the unit with the provided id
  - Syntax: **`supplement AggressionInhibitor sharo`** – adds an AggressionInhibitor to the Dog sharo (created in the previous examples)
- **Status** command – prints information about all units in the HoldingPen
  - Syntax: **`status`**
  - The base class Unit overrides the **ToString()** method appropriately, describing its state. You need not concern yourself with this operation
  - Note: The **ToString()** command prints the object class name and its supplements' class names – be sure when you are creating units and supplements to use the names exactly as they are described below

Tasks

You are tasked with extending the API by implementing several commands and object types. You are **not allowed to edit any existing class from the original code of the API**. You **are NOT allowed to edit the Main method**. You are **only allowed to edit the InitializePen()** method in the **Program** class.

- **Catalists** are Supplements, which improve the Health, Power and Aggression of a unit. Implement:
  - A **`PowerCatalyst`** – has a **`PowerEffect`** of **3**
  - A **`HealthCatalyst`** – has a **`HealthEffect`** of **3**
  - An **`AggressionCatalyst`** – has an **`AggressionEffect`** of **3**
- Implement a **`Tank`**
  - A **`Tank`** is a type of **`Unit`**
  - The **`Tank`** has a base **`Power`** of **25**, a base **`Health`** of **20**, and a base **`Aggression`** of **25**
  - The **`Tank`** is classified as a **`Mechanical`** Unit.
- Implement a **`Marine`**
  - The **`Marine`** is a type of **`Human`**
  - It has the same base **`Power`**, **`Health`**, **`Aggression`**
  - It has a supplement by default – **`WeaponrySkill`**
    - The **`WeaponrySkill`** does not directly affect any of the properties of the **`Marine`**

        - The **WeaponrySkill** cannot be added with the supplement command (This means you cannot create it through the console, it doesn't mean you can't use the **AddSupplement** method)
    - When a **Marine** attacks, it always picks a target, such that:
        - The target's **Power** is less than or equal to the Marine's **Aggression**
        - If there is more than one such target, the marine picks the one with the highest **Health**
- Implement a **Weapon** Supplement
    - A **Weapon** is a Supplement, which increases the **Power** of a **Unit** by **10** and its **Aggression** by 3, but only if the **Unit** already has a **WeaponrySkill** Supplement. If not, the **Weapon** Supplement does not have any effect.
- Implement an **InfestationSpores** Supplement
    - The **InfestationSpores** Supplement has an **AggressionEffect** of **20** and a **PowerEffect** of **-1**
    - The **InfestationSpores** Supplement does not accumulate like the other Supplements – even if two or more Infestations are added, the total **AggressionEffect** stays **20** (the same goes for **PowerEffect**)
    - The **InfestationSpores** Supplement cannot be added with the supplement command
- Implement a **Parasite**
    - The **Parasite** is a type of **Unit**, which can Infest
        - Infesting is equivalent to adding an **InfestationSpores** Supplement to the target
    - The **Parasite** is classified as a Biological Unit.
    - The **Parasite** has all base values set to 1
    - When a **Parasite** is offered to Interact, it always tries to find a Unit to infest
        - The target **Unit** can be any unit different than itself
        - If there are multiple such units, the **Parasite** picks the one with the least **Health**
- Implement a **Queen**
    - The **Queen** is a type of **Unit**, which can infest
        - Infesting is equivalent to adding an **InfestationSpores** Supplement to the target
    - The **Queen** has a base **Health** of **30** and all of its other base values are set to **1**
    - The **Queen** is classified as a **Psionic** Unit
    - The **Queen** interacts in the same way as the **Parasite**
- **Infesting has some requirements**
    - A **Biological** unit can only be infested by another **Biological** unit
    - A **Mechanical** unit can only be infested by a **Psionic** unit
    - A **Psionic** unit can only be infested by another **Psionic** unit
    - There is some code in the API reflecting these rules, seek it out

Input and Output Data

You should not concern yourself with handling input and output data – the engine does it for you. You should only consider how to implement the required commands. See the existing API code for hints. Also:

- The names in the commands will always consist of upper and lowercase English letters only.
- If for some reason a command is illegal, just skip it
- The **ReactTo** method needs to be implemented specifically for ONLY two of the above tasks – most supplements don't need to react (at least in this task)

Sample Input	Sample Output
insert Dog Sharo insert Parasite Paro proceed status insert Tank Tanio proceed status insert Queen Murphy proceed status insert Marine Marin supplement Weapon Marin supplement AggressionCatalyst Marin supplement AggressionCatalyst Marin proceed status end	Dog Sharo (Biological) [Health: 4, Power: 4, Aggression: 22, Supplements: [InfestationSpores]] Tank Tanio (Mechanical) [Health: 20, Power: 25, Aggression: 25, Supplements: []] Tank Tanio (Mechanical) [Health: 20, Power: 24, Aggression: 45, Supplements: [InfestationSpores]] Queen Murphy (Psionic) [Health: 5, Power: 1, Aggression: 1, Supplements: []] Tank Tanio (Mechanical) [Health: 20, Power: 24, Aggression: 45, Supplements: [InfestationSpores, InfestationSpores]] Marine Marin (Biological) [Health: 10, Power: 14, Aggression: 10, Supplements: [WeaponrySkill, Weapon, AggressionCatalyst, AggressionCatalyst]]