Object-Oriented Programming – Practical Exam

Problem 2 – Role-Playing Game API

You are given a role-playing game API that gives the base for creating a game with characters, resources, battles and resource gathering. The API includes an engine, which executes controls the game world and executes commands on the objects within it. You are also given a C# file, which has a Main method and uses the API for processing commands from the input.

There are some simple rules the API supports:

- Objects can be created anywhere
- Objects can go anywhere if they support the "go" command
- Objects can attack if they are fighters
    - Attacks always have single targets – one object can attack at most one other object at a time
- Objects can gather resources if they are gatherers
- Objects can be gathered if they are resources
- Objects have owners. An owner can be interpreted as "the player to which the object belongs"
- An object can be neutral, meaning "belong to player 0".

The following lines will help you better understand the API.

Important Classes and Interfaces

These are the API's interfaces:

- **`IWorldObject`** – provides methods for getting an object's **`HitPoints`** (life remaining), **`Position`** (x and y coordinates, and checking whether the object **`IsDestroyed`**.
- **`IResource`** – provides methods to get resource values from an object – **`Quantity`** and **`Type.`**
- **`IControllable`** – provides a **`Name`** property, enabling addressing a unit to give it orders.
- **`IFighter`** – provides **`AttackPoints`** and **`DefensePoints`** properties, representing how powerful an object is in battle. Also provides a method for picking one target from a multitude of available ones – **`GetTargetIndex(List<WorldObject>)`**. If the **object does not find a suitable target from the given list, GetTargetIndex should return -1**.
- **`IGatherer`** – provides a **`TryGather(IResource`**) method, which returns true if the gatherer successfully gathers a resource. Gathering a resource is always complete – a resource cannot be gathered partially (e.g. you cannot gather half a tree).

These are the API's classes and structures:

- **`Point`** – **`struct`**, represents a **two-dimensional point with integer coordinates** and provides a **`Parse(string)`** and overloaded **`ToString()`** method.
- **`WorldObject`** – **`abstract`** implementation of the **`IWorldObject`** interface. Has an additional **integer** property **`Owner`**. It is used to determine "on which side" the object is – **1** means the **first player**, **2** means the **second player**, **3** means the **third player** and so on. Zero (**0**) means the object is "neutral".
- **`MovingObject`** – **`abstract`** base for **objects that can change their position**, through the "go" command (implemented with the **`GoTo(Point)`** method). All such objects **must** inherit the **`MovingObject`** class.

- **`StaticObject`** – **`abstract`** base for objects that never change their position. All such objects **must** inherit the **`StaticObject`** class.
- **`Character`** – **`abstract`**, inherits the **`MovingObject`** and implements the **`IControllable`** interface. Should be used for any controllable characters.
- **`Lumberjack`** – implementation the **`IGatherer`** interface, inherits **`Character`**. Represents a lumberjack, which can "gather" (chop down) trees.
- **`Guard`** – implementation of the **`IFighter`** interface, inherits **`Character`**. Represents a guard, which has attack and defense capabilities.
- **`Tree`** – implementation of the **`IResource`** interface, inherits **`StaticObject`**. Represents a tree which has a **`Size`**, related to the **`Quantity`** property of the **`IResource`** interface and a **`Type`** of **`Lumber.`**
- **`Engine`** – handles commands and executes them on game world, keeps several lists of the objects in it, provides methods for adding objects and removes IsDestroyed objects after each command.

Commands

There are two types of commands the Engine supports:

- Object creation command – creates an object in the world
  - Format: **create <object-type-name> <object-parameters…>**
  - Example: **create tree 10 (0,0)** – creates a tree with a size of 10 at the position (0, 0)
  - Example: **create guard Jack (0,0) 1** – creates guard named Jack, owned by player 1, at (0, 0)
- Object instance command – orders an object to execute a command. If the object can execute such a command, a string is printed, describing the result of the command. If not, a string is printed, notifying the inability of the object to execute the command.
  - Format: **<object-name> <command-name> <command-parameters>**
  - Commands:
    - **`attack`** `(no parameters)`
    - **`gather`** `(no parameters)`
    - **`go <position>`**
  - Example: **Joro attack** – causes the Engine to search for a **`IControllable`** object with the **`Name`** Joro and attempt to use that object as an **`IFighter`**, to attack other objects at the object's coordinates, printing the corresponding strings for success or failure.
  - Example: **Joro gather** - causes the Engine to search for a **`IControllable`** object with the **`Name`** Joro and attempt to use that object as an **`IGatherer`**, to gather the resource at the object's coordinates, printing the corresponding strings for success or failure.
  - Example: **Joro go (42,53)** – causes the Engine to search for a **`IControllable`** object with the **`Name`** Joro and call that object's **`GoTo(Point)`** method with a **`Point`** with coordinates (42, 53).

Here is a list of all commands the Engine supports currently:

- **create tree <size> <position>** - creates a tree with the specified size at the specified coordinates
- **create lumberjack <name> <position> <owner>** - creates a lumberjack with the specified name at the specified coordinates and belonging to the specified owner (player)
- **create guard <name> coordinates> <owner>** - same as the previous, but creates a guard
- **<name> go <position>** – makes the object with the specified name go to the specified coordinates

- **<name> attack** – orders the object with the specified name to attack another object at its coordinates
- **<name> gather –** orders the object with the specified name to gather a resource at its coordinates

The Engine handles all listed commands and **you shouldn't write your own parsing code** for these commands.

Study the aforementioned classes to get a better understanding of how they.

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 allowed** to edit the **`GetEngineInstance()`** method.

- Implement a command to create a **`Knight`**. The **`Knight`** should be **able to move** and to **execute attack commands**. The **`Knight`** should have 100 **`AttackPoints`**, 100 **`DefensePoints`** and 100 **`HitPoints`**. The knight should have an **owner, name, and position**. When attacking, the **`Knight`** should always pick the **first available target to attack**, which is **not neutral** and does **not belong to the same player** as the **`Knight`.**
  - Format: create knight <name> <position> <owner> - works the same way as the create guard command, but instead creates a **`Knight`**.
  - Example: create knight Joro (1,1) 1 – creates a Knight with the name Joro, at coordinates (1, 1) and belonging to player 1.
- Implement a command to create a **`House`**. The **`House`** should **not be able to move**. The house should have 500 **`HitPoints`**. The house should have a **position and an owner**.
  - Format: create house <position> <owner> - creates a house at the specified position, belonging to the specified player
  - Example: create house (-1,-1) 2 – creates a house at coordinates (-1, -1), belonging to player 2
- Implement a command to create a **`Giant`**. The **`Giant`** should be **able to move** and to **execute attack commands**. It should have 150 **`AttackPoints`**, 80 **`DefensePoints`** and 200 **`HitPoints`**. It should have a **name and a position**, but should be **always neutral**. The Giant should also be **able to gather Stone resources**. When a **`Giant`** gathers such a resource, his **`AttackPoints`** are **permanently increased by 100**. This should **only work once**. When **attacking**, the **`Giant`** should pick the **first available target**, **which is not neutral**.
  - Format: create giant <name> <position> - creates a giant which is at the specified position with the specified name, and is neutral (i.e. has the owner 0)
  - Example: create giant BigGuy (10,0) – creates a giant with the name BigGuy at the coordinates (10, 0), belonging to player 0
- Implement a command to create a **`Rock`**. The **`Rock`** should **not be able to move**. The **command should set** the **`HitPoints`** of the **`Rock`**. The **`Rock`** should be a **resource** with a **`Type`** property equal to **`Stone`**. The **`Quantity`** of the Rock should be **half** it's **`HitPoints`.** The **`Rock`** should **always be neutral** and have a **position**.
  - Format: create rock <hitpoints> <position> - creates a rock at the specified position, which has the specified hitpoints and is neutral
  - Example: create rock 50 (7,-3) – creates a neutral rock with 50 hit points at the coordinates (7,-3).
- Implement a command to create a **`Ninja`**. The **`Ninja`** should be **able to move** and to **execute attack commands**. The Ninja should have 1 **`HitPoints`**, but should be **invulnerable** – it should not be able to be destroyed by other objects. The **`Ninja`** should **initially have no `AttackPoints`**. The **`Ninja`** should be **able to gather stone and lumber resources**. For **each lumber resource** the **`Ninja`** has gathered, its

AttackPoints should increase by the resource's quantity. For **each stone resource** the **Ninja** has gathered, its **AttackPoints should increase by the resource's quantity multiplied by 2**. The **Ninja should have an owner**, specified by the command. The Ninja should have a **name and position**. The **Ninja** should always **attack the target**, which is **not neutral**, does **not belong to the same player**, and has the **highest HitPoints** of all the available targets.

- Format: create ninja <name> <position> <owner> - creates a ninja with the specified name, at the specified position, which belongs to the specified player.
- Example: create ninja Joro (0,0) 1 – creates a ninja with the name Joro, at the coordinates (0,0), belonging to player 1

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 creation commands. See the existing Engine code for hints. Also:

- The names in the commands will always consist of upper and lowercase English letters only. The numbers in the commands will always be 32-bit signed integers (System.Int32).
- There will never be two resources at the same position or two objects with the same name
- There will never be a neutral **Guard**, **Knight**, **Lumberjack** or **Ninja**

Sample Input	Sample Output
create guard Peter (0,0) 1 create lumberjack Jack (1,1) 1 create tree 10 (2,2) Jack go (2,2) Jack gather create guard Max (0,0) 1 Max attack create guard Evil (0,0) 2 Peter attack Max attack Peter attack Peter attack create knight EvilKnight (0,0) 2 EvilKnight attack EvilKnight attack EvilKnight attack EvilKnight attack EvilKnight go (2,2) EvilKnight attack create giant cecameca (10,10) create rock 6 (10,10) cecameca gather cecameca go (2,2) cecameca attack end	Lumberjack Jack is now at position (2,2) Lumberjack Jack gathered 10 Lumber from Tree No targets to attack at Guard Max's position Guard Peter attacked Guard Evil and did 30 damage Guard Max attacked Guard Evil and did 30 damage Guard Peter attacked Guard Evil and did 30 damage Guard Peter attacked Guard Evil and did 10 damage Knight EvilKnight attacked Guard Peter and did 80 damage Knight EvilKnight attacked Guard Peter and did 20 damage Knight EvilKnight attacked Guard Max and did 80 damage Knight EvilKnight attacked Guard Max and did 20 damage Knight EvilKnight is now at position (2,2) Knight EvilKnight attacked Lumberjack Jack and did 50 damage Giant cecameca gathered 3 Stone from Rock Giant cecameca is now at position (2,2) Giant cecameca attacked Knight EvilKnight and did 100 damage