# OOP Exam - ISIS

ISIS is a militant group which fights against the world. Write a C# or Java application that simulates their military life.

## Task 1 - Implement the class hierarchy

The application will support creating militant groups and fights between them.

A group has a **name**, **health** and **damage**. It also has a **war effect**. A war effect is triggered when a militant group falls to **less or equal** to **half its initial health**. The following war effect should be supported:

* **Jihad** - doubles the group’s damage. Each consecutive day the group loses **5 damage points**. The damage cannot fall below its initial value (the damage before the effect was toggled).
* **Kamikaze** - The group gains **50 health points**. Each consecutive day the group loses **10 health points**.

An effect can only be triggered **once**. It should be triggered even if the group falls to 0 health points. If it is triggered a second time, an error should be raised.

Groups can attack each other with predefined attack types

* **Paris** - the group produces an attack with **damage** equal to its **own damage**
* **SU24** - the group loses **half its current health** (e.g. from 69 health loses 34 health = 35 health left) and produces an attack with **damage** equal to **twice its own damage**
  + The group cannot fall below 1 health points from attacking with **SU-24**

A militant group can perform an attack multiple times. It can have only a **single attack** (either Paris, SU-24 or any other supported attack) and a **single war effect** (either Jihad, Kamikaze or any other supported effect).

### Other Notes

* If a group’s attack **triggers an effect**, the effect should be applied **immediately** (i.e. an **effect triggered by an attack** can affect the **attack** that triggered it)
* A group should not fall below **0 health**
* **Dead groups** cannot attack / be attacked

## Task 2 – Validation rules and clean code

Encapsulate all internal behavior. The implemented classes should not reveal any internal logic.

Avoid code repetition and promote code reusability by applying the good practices of OOP.

## Task 3 – Application Logic

From the standard input you will receive **commands**, each on a separate line. The application should support the following commands:

* **<name>.create(<health>, <damage>, <war effect>, <attack>)** - adds a new group with the specified war effect and attack
* **<attacker>.attack(<target>)** - forces a group to perform an attack on another group

The **attacking group** produces an **attack** that deals damage to the **target group’s health**.

* **world.akbar()** - does nothing, skips the day and the world lives and peace today
* **world.status()** - prints data about the current state of the application in the following format:

|  |
| --- |
| **Group {name}: {health} HP, {damage} Damage**  **...** |

Groups should be printed in order of health left from largest to smallest, if two groups have identical health, then they should be compared by their damage from largest to smallest, if these statistics are also equal, then in order of appearance the game.

If a group has been killed, the format should instead be:

|  |
| --- |
| **Group {name} AMEN** |

* **world.apocalypse()** – the world comes to an end and so does the program.

Each command should progress the world with **1 day** after it is executed.

## Task 4 – Loose Coupling

The application should support the creation of groups with **any war effect** and **attack**.

## Task 5 – Input / Output Independence

The application should be designed to work with **any input source** and **output destination**. In other words, it should **NOT** depend on the console.

## \* Bonus Task 6 – Military Group Events

Implement a fifth command:

* **world().getEvents().status()** - if passed as **first command** in input the application should **print detailed information** when groups attack each other:
* When a group toggles its effect

|  |
| --- |
| **Group {name} toggled {effect-type}** |

* When a group is killed (its health drops to 0 after all effects are taken into consideration)

|  |
| --- |
| **Group {name} was killed** |

The groups should **NOT** directly interact with the application logic core and input/output sources.

This task is not part of the automated tests in the Judge system.

## Input

The input will be read from the standard input. On each line a command will be given (one of the described above).

Commands can be chained like world.status().akbar().status(). Chaining can be performed only on compatible objects. E.g. if world is the object, only commands compatible to the world can be chained.

## Output

The output should be printed on the console. Upon receiving the **world**.status() command, print the current status of the world as described above.

## Constraints

* The **health** and **damage** will be valid 32-bit integer numbers
* The input will always end with the world.apocalypse() command
* The **world().getEvents().status()** command will always come first if present in the input
* There will always be compatible commands to the given object

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Cenko.create(30, 15, Kamikaze, Paris)  Boko.create(50, 20, Jihad, SU24)  Boko.attack(Cenko)  world.status()  world.status()  world.status()  world.status()  world.status()  world.status()  world.apocalypse() | Group Cenko: 50 HP, 15 Damage  Group Boko: 25 HP, 40 Damage  Group Cenko: 40 HP, 15 Damage  Group Boko: 25 HP, 35 Damage  Group Cenko: 30 HP, 15 Damage  Group Boko: 25 HP, 30 Damage  Group Boko: 25 HP, 25 Damage  Group Cenko: 20 HP, 15 Damage  Group Boko: 25 HP, 20 Damage  Group Cenko: 10 HP, 15 Damage  Group Boko: 25 HP, 20 Damage  Group Cenko AMEN |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Fiki.create(90, 5, Kamikaze, SU24)  Jorjo.create(30, 25, Kamikaze, SU24)  Fiki.attack(Jorjo)  world.status()  Fiki.attack(Jorjo)  world.status()  world.apocalypse() | Group Fiki: 95 HP, 5 Damage  Group Jorjo: 20 HP, 25 Damage  Group Jorjo: 60 HP, 25 Damage  Group Fiki: 33 HP, 5 Damage |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sir.create(70, 20, Jihad, SU24)  Stenly.create(33, 15, Jihad, SU24)  Royce.create(50, 20, Kamikaze, SU24)  world.status()  Stenly.attack(Royce)  world.status()  world.status()  world.apocalypse() | Group Sir: 70 HP, 20 Damage  Group Royce: 50 HP, 20 Damage  Group Stenly: 33 HP, 15 Damage  Group Sir: 70 HP, 20 Damage  Group Royce: 70 HP, 20 Damage  Group Stenly: 17 HP, 15 Damage  Group Sir: 70 HP, 20 Damage  Group Royce: 60 HP, 20 Damage  Group Stenly: 17 HP, 15 Damage |

|  |  |
| --- | --- |
| **Input** | **Output** |
| world().getEvents().status()  Petya.create(20, 10, Jihad, Paris)  Emi.create(30, 15, Kamikaze, Paris)  Petya.attack(Emi).attack(Emi)  Emi.attack(Petya)  Emi.attack(Petya)  world.akbar()  world.status()  world.apocalypse() | Group Emi toggled Kamikaze  Group Petya toggled Jihad  Group Petya was killed  Group Emi: 30 HP, 15 Damage  Group Petya AMEN |