OOP Exam - Blobs

Blobs - slimy little creatures who have been at war for the last 300 years. Write a C# application which supports creating blobs and simulating fights between them.

Task 1 - Implement the Game Objects

A blob has a **name**, **health** and **damage**.

A blob also has a **behavior**. A behavior is triggered when a blob falls to **less or equal** to **half its initial health**. The following behaviors should be supported:

- **Aggressive Behavior** - doubles the blob's damage. Each consecutive turn the blob loses **5 damage**. The unit's damage cannot fall below its initial value (the damage before the behavior was toggled).
- **Inflated Behavior** - The blob gains **50 health**. Each consecutive turns the blob loses **10 health**.

A behavior can only be triggered **once**. It should be triggered even if the blob falls to 0 health. If it is triggered a second time, an error should be raised.

A blob can **attack** another blob. The following attacks should be supported:

- **Putrid Fart** - the blob produces an attack with **damage** equal to its **own damage**
- **Blobplode** - the blob loses **half its current health** (e.g. from 55 health loses 27 health = 28 health left) and produces an attack with **damage** equal to **double its own damage**
    - The blob cannot fall below 1 health from attacking with Blobplode

A blob can perform an attack multiple times (only once per turn). A blob can have only a **single attack** (either Putrid Fart, Blobplode or any other attack) and a single behavior (either Aggressive, Inflated or any other behavior).

Other Notes

- If a blob's attack **triggers a behavior**, the behavior should be applied **immediately** (i.e. a **behavior triggered by an attack** can affect the **attack** that triggered it)
- A blob should not fall below **0 health**
- **Dead blobs** cannot attack / be attacked

Task 2 - Flexible Blobs

Design the blobs so they can work flexibly with **any behavior** and **any attack**.

Task 3 - Improve the Models

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 4 - Application Logic

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

- `create <name> <health> <damage> <behavior> <attack>` - adds a new blob with the specified behavior and attack

- **`attack <attacker> <target>`** - forces a blob to perform an attack on another blob

The **attacking blob** produces an **attack** that deals damage to the **target blob's health**.

- **`pass`** - does nothing, skips the turn and progresses the game
- **`status`** - prints data about the current state of the game in the following format:

```
Blob {name}: {health} HP, {damage} Damage
...
```

Blobs should be printed in order of entry in the game.

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

```
Blob {name} KILLED
```

- **`drop`** - ends the program

Each command should progress the game with **1 turn** after it is executed.

Task 5 - Loose Coupling

The application should support the creation of blobs with **any behavior** and **attack**.

Task 6 - 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 7 - Blob Events

Implement a fifth command:

- **`report-events`** - if passed as **first command** in input the engine should **print detailed information** when blobs attack each other:
  - When a blob toggles its behavior

    ```
    Blob {name} toggled {behavior-type}
    ```

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

    ```
    Blob {name} was killed
    ```

The blobs should **NOT** directly interact with the engine or any input/output classes.

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).

Output

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

Constraints

- The **health** and **damage** will be valid 32-bit integer numbers
- The input will always end with the **`drop`** command
- The **`report-events`** command will always come first if present in the input

Examples

Input	Output
create Cenko 30 15 Inflated PutridFart create Boko 50 20 Aggressive Blobplode attack Boko Cenko status status status status status status drop	Blob Cenko: 50 HP, 15 Damage Blob Boko: 25 HP, 40 Damage Blob Cenko: 40 HP, 15 Damage Blob Boko: 25 HP, 35 Damage Blob Cenko: 30 HP, 15 Damage Blob Boko: 25 HP, 30 Damage Blob Cenko: 20 HP, 15 Damage Blob Boko: 25 HP, 25 Damage Blob Cenko: 10 HP, 15 Damage Blob Boko: 25 HP, 20 Damage Blob Cenko KILLED Blob Boko: 25 HP, 20 Damage

Input	Output
create Fiki 90 5 Inflated Blobplode create Jorjo 30 25 Inflated Blobplode attack Fiki Jorjo status attack Fiki Jorjo status drop	Blob Fiki: 95 HP, 5 Damage Blob Jorjo: 20 HP, 25 Damage Blob Fiki: 33 HP, 5 Damage Blob Jorjo: 60 HP, 25 Damage

Input	Output
create Sir 70 20 Aggressive Blobplode create Stenly 33 15 Aggressive Blobplode create Royce 50 20 Inflated Blobplode status attack Stenly Royce status status drop	Blob Sir: 70 HP, 20 Damage Blob Stenly: 33 HP, 15 Damage Blob Royce: 50 HP, 20 Damage Blob Sir: 70 HP, 20 Damage Blob Stenly: 17 HP, 15 Damage Blob Royce: 70 HP, 20 Damage Blob Sir: 70 HP, 20 Damage Blob Stenly: 17 HP, 15 Damage Blob Royce: 60 HP, 20 Damage

Input	Output
report-events create Petya 20 10 Aggressive PutridFart create Emi 30 15 Inflated PutridFart attack Petya Emi attack Petya Emi attack Emi Petya attack Emi Petya pass status drop	Blob Emi toggled InflatedBehavior Blob Petya toggled AggressiveBehavior Blob Petya was killed Blob Petya KILLED Blob Emi: 30 HP, 15 Damage