# AirCombat

### Tasks

You are given an almost working application which simulates **AirCraft** battles.

Use the problem description to get familiar with the project and what it is expected to do.

#### Task 1: Refactor Part class – 25 pts

Refactor Part class to follow the OOP principles.

#### Task 2: Find and Fix Bugs – 25 pts

Find all business logic bugs and fix them.

You can use the **input** and **output** sections to check the business logic validity.

#### Task 3: Command Pattern – 50 pts

Implement the [command pattern](https://www.dofactory.com/net/command-design-pattern).

### Structure

The structure of the software circles around AirCrafts and Parts.

#### AirCrafts

The AirCrafts are initialized with:

* Model – a **string**.
* Weight – a **floating-point number**.
* Price – a **decimal number**.
* Attack – an **integer**.
* Defense – an **integer**.
* HitPoints – an **integer**.

There are 2 types of AirCrafts - Bastilon and Enforcer.

#### Parts

The Parts are initialized with:

* Model – a **string**.
* Weight – a **floating-point number**.
* Price – a **decimal number**.

There are generally 3 types of Parts.

##### ArsenalPart

The ArsenalPart is initialized with an additional property:

* AttackModifier – an **integer**.

##### ShellPart

The ShellPart is initialized with an additional property:

* DefenseModifier – an **integer**.

##### EndurancePart

The EndurancePart is initialized with an additional property:

* HitPointsModifier – an **integer**.

#### Assembler

The Assembler contains 3 collections – 1 for the **ArsenalParts**,1 for the **ShellParts**, and 1 for the **EnduranceParts**.

The class exposes **3 methods** for adding Parts – one for the **ArsenalParts,** one for the **ShellParts**, and one for the **EnduranceParts**.

The class also exposes **3 methods** for **extracting** the **total stat modification** each type of parts gives to the **AirCraft**.

#### BattleOperator

The BattleOperator class exposes **1 method** for **handling Battles** – the method **accepts 2 AirCraft**s and initiates a Battle between them, ultimately **resulting** in a **winner**. The winner’s model is **returned** as **result** of the **method**.

The 2 AirCrafts fight in turns with the **first given AirCraft** being the **first 1** to **attack**.

**First**, the **attacker attacks**, **then**, the **target attacks back**. Each turn they lose **hitPoints**, due to the attack, by the following formula:

receivingAirCraftHitPoints -= (attackingAirCraftAttack - (receivingAirCraftDefense + (receivingAirCraftWeight / 2)))

As you see the **Defense** and **Weight** of the receivingAirCraft **reduce** the **attack damage** of the attackingAirCraft, which is a normal tactic.

The process of exchanging attacks continues, until one’s **hitPoints** is **lower than** or **equal** to **0**.

### Functionality

The functionality of the software involves adding AirCrafts, adding Parts to the AirCrafts, and so on. As you see the AirCrafts and Parts are the main entities of the program. The model is the **property** that will **identify** them. The **model** will also, always be **unique** in the input.

In **some** of the **commands**, you’ll receive models which may refer to a AirCraft and a Part. You must check what is the object with that model, and process the command depending on the result.

Each AirCraft has an Assembler, in which it **stores** its Parts.   
The business logic of the program involves: adding AirCrafts and parts, inspecting AirCrafts and parts, fighting between AirCrafts.

Check below, each section, and the functionality it describes.

#### AirCrafts

The AirCrafts are the main actors in the business logic. They have **stats** which **define** their **power**. Those **stats** can be **upgraded** by **adding parts** to them, which is done through the Assembler.

**Battles** are triggered **between 2 AirCrafts**. The **resulting winner** of the battle, should **stay** in the data, while the loser should be **removed**.

Battles are controlled by the BattleOperator. When 2 AirCrafts are passed to the BattleOperator, it **returns** the model of the **winning AirCraft**. You should consider that in your logic.

#### Parts

The Parts have no business logic around themselves. They are just **data models**.

#### Commands

There are several commands which are given from the user input, in order to control the program.   
Here you can see how they are formed.

The **parameters** will be given in the **EXACT ORDER**, as the one **specified below**.   
You can see the exact input format in the **Input section**.

**Each** **command** will **generate an output** **result**, which you must **print**.  
You can see the exact output format in the **Output section**.

##### AirCraft Command

**Parameters** – **type** (string), **model** (string), **weight** (double), **price** (decimal), **attack** (integer), **defense** (integer), **hitPoints** (integer).

Creates a AirCraft of the **given type**, with the **given model**.   
The type will either be “Bastilon” or “Enforcer”.

##### Part Command

**Parameters** – **AirCraftModel** (string), **type** (string), **model** (string), **weight** (double), **price** (decimal), **additionalParameter** (integer).

Creates a Part of the **given type** with the **given model** and **adds** it to the Assembler of the **AirCraft** with the **given AirCraftModel**.

The type will either be “Arsenal”, “Shell” or “Endurance”.

Depending on the Part type, the additionalParameter will be set to a different property:

* If it’s an ArsenalPart the **additionalParameter** will be set ot the attackModifier.
* If it’s a ShellPart the **additionalParameter** will be set ot the defenseModifier.
* If it’s an EndurancePart the **additionalParameter** will be set ot the hitPointsModifier.

##### Inspect Command

**Parameters** – **model** (string)

Brings data about the **AirCraft** or the **Part** with the **given model**, providing **detailed** **information** about it.

##### Battle Command

**Parameters** – **AirCraft1Model** (string), **AirCraft2Model** (string)

Initiates a battle between **2 AirCrafts**. You should **pass** the **2 parameters** to the BattleOperator, and when you get the **resulting winner**, you should **remove** the **loser** from the **data**.

##### Terminate

**Exits** the program. Prints **detailed information** about the **current state** of the system.

### Input

The input consists of several commands which will be given in the format, specified below: :

* AirCraft {AirCraftType} {model} {weight} {price} {attack} {defense} {hitPoints}
* Part {AirCraftModel} {partType} {model} {weight} {price} {additionalParameter}
* Inspect {model}
* Battle {AirCraft1Model} {AirCraft2Model}
* Terminate

### Output

Each of the commands generates **output**. Here are the **output formats** of each command:

* AirCraft Command – creates a AirCraft of the given type, with the given model. Prints the following result:

**Created {type} AirCraft – {model}**

* Part Command – adds a Part of the given type, with the given model to a specified AirCraft.

**Added {partType} - {partModel} to AirCraft - {AirCraftModel}**

* Inspect command – provides **detailed** **information** about a **AirCraft** or a **Part**, in one of the following formats:

|  |  |
| --- | --- |
| AirCraft | Part |
| {AirCraftType} – {AirCraftModel}  Total Weight: {totalWeight}  Total Price: {totalPrice}  Attack: {totalAttack}  Defense: {totalDefense}  HitPoints: {totalHitPoints}  Parts: {part1Model}, {part2Model}... | {partType} Part – {partModel}  +{additionalParamValue} {additionalParam} |

Because of the fact, that the **Part** is not particular, the additionalParameter should either be “**Attack**”, “**Defense**”, “**HitPoints**”.

In case **there** **are no Parts** for the AirCraft, print “Parts: None”.

The totalWeight and totalPrice must be printed to the **3rd digit** **after** the **decimal point**.

* + The **Parts** in the output should be **ordered** by **order** of **input**.
* Battle command – The command should return as output the winner in the following format:

**{AirCraft1Model} versus {AirCraft2Model} -> {winnerModel} Wins! Flawless Victory!**

* Terminate command – Terminates the program; **prints** detailed statistics about the whole system. The format, in which the statistics should be printed is:

Remaining AirCrafts: {AirCraft1Model}, {AirCraft2Model}...  
Defeated AirCrafts: {defeatedAirCraft1Model}, {defeatedAirCraft2Model}...  
Currently Used Parts: {countOfCurrentlyUsedParts}

* + Remaining AirCrafts are all AirCrafts that **have not been** defeated in a battle.
  + Defeated AirCrafts are all AirCrafts that **have been** defeated in a battle.
  + Currently Used Parts is the **amount** of **parts** used by the Remaining AirCrafts. (Exclude those from the Defeated AirCrafts).
  + In case there are no Remaining AirCrafts or Defeated AirCrafts print “None”.
  + **All data** in the output should be **ordered** by **order** of **input**.

### Constrains

* All **integers** in the input will be in **range [0, 800.000.000]**.
* All **floating-point numbers** in the input will be in **range [0, 1.000.000.000]**.
* All **strings** in the input may contain **any ASCII character**, except **space** (‘ ‘).
* All **input lines** will be **absolutely valid**.
* There will be **no** non-existent **models** or **types** in the input.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| AirCraft Bastilon SA-203 100 300 1000 450 2000  AirCraft Enforcer AKU 1000 1000 1000 1000 1000  Part SA-203 Arsenal Cannon-KA2 300 500 450  Part AKU Shell Shields-PI1 200 1000 750  Inspect SA-203  Inspect AKU  Terminate | Created Bastilon AirCraft - SA-203  Created Enforcer AirCraft - AKU  Added Arsenal - Cannon-KA2 to AirCraft - SA-203  Added Shell - Shields-PI1 to AirCraft - AKU  Bastilon - SA-203  Total Weight: 400.000  Total Price: 800.000  Attack: 1450  Defense: 450  HitPoints: 2000  Parts: Cannon-KA2  Enforcer - AKU  Total Weight: 1200.000  Total Price: 2000.000  Attack: 1000  Defense: 1750  HitPoints: 1000  Parts: Shields-PI1  Remaining AirCrafts: SA-203, AKU  Defeated AirCrafts: None  Currently Used Parts: 2 |
| AirCraft Enforcer Destroyer-2U 1500 100000 9500 5000 15000  AirCraft Enforcer Falcon-303 750 55000 4500 2000 6500  AirCraft Bastilon Rhino-CE 3000 85000 2000 4000 20000  Part Destroyer-2U Arsenal Cannon-X 1000 50000 5000  Part Destroyer-2U Arsenal Cannon-Y 1000 50000 5000  Part Rhino-CE Shell Shield-EX 2000 45000 3000  Battle Destroyer-2U Rhino-CE  Inspect Destroyer-2U  Terminate | Created Enforcer AirCraft - Destroyer-2U  Created Enforcer AirCraft - Falcon-303  Created Bastilon AirCraft - Rhino-CE  Added Arsenal - Cannon-X to AirCraft - Destroyer-2U  Added Arsenal - Cannon-Y to AirCraft - Destroyer-2U  Added Shell - Shield-EX to AirCraft - Rhino-CE  Destroyer-2U versus Rhino-CE -> Destroyer-2U Wins! Flawless Victory!  Enforcer - Destroyer-2U  Total Weight: 3500.000  Total Price: 200000.000  Attack: 19500  Defense: 5000  HitPoints: 15000  Parts: Cannon-X, Cannon-Y  Remaining AirCrafts: Destroyer-2U, Falcon-303  Defeated AirCrafts: Rhino-CE  Currently Used Parts: 2 |