# OOP Advanced Exam - Recycling Station

Waste collection and recycling is big problem for modern societies, as the consumption of goods increase so does the produced waste, thus the need for a fast and efficient way to deal with it is needed. You, as a young and promising programmer, have been tasked with the job to create the software for a recycling station. Since creating software from zero is not an easy task, you found the source code for a framework on the internet to help you.

### Core Logic

Your task is to create the software for a Recycling Station, it should keep information about its resources (Energy and Capital), should support operations for processing garbage, printing its current state and should also be able to deny processing certain garbage types based on a management requirement.

**Models**

Your program should support three types of garbage - **Recyclable garbage, Burnable garbage** and **Storable garbage.**   
Each garbage has a **Name**, a **Weight** (in kilograms) and **VolumePerKg** (in cubic centimeters per kg).

Each garbage is processed in a different way and produces different results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Energy Produced | Energy Used | Revenue | Cost |
| Recyclable garbage | 0 | 50% of total garbage volume | 400 \* garbage weight | 0 |
| Burnable garbage | 100% of total garbage volume | 20% of total garbage volume | 0 | 0 |
| Storable garbage | 0 | 13% of total garbage volume | 0 | 65% of total garbage volume |

**Total garbage volume = garbage weight \* garbage volume per kg**

**Management**

Your program should also be able to deny processing a specific type of garbage when the total **EnergyBalance** or **CapitalBalance** of the Recycling Station falls **under** certain ammounts. During operation of your program new management requirments can be received that should **REPLACE** the previous management requirment. By default your program should start **without** a management requirement (i.e. all garbage types are accepted).

### Framework Overview

The framework receives and maps strategies for disposal of garbage to attributes inherited from an abstract attribute class it exposes, then when receiving a waste object it searches for a strategy to process it, based on its attribute.

The framework has the following classes:

**Attributes**

1. **[DisposableAttribute] -** An abstract base class for all attributes that specify the disposal strategy. The provided framework requires the passed in attributes to inherit from this one.

**Interfaces**

1. **IWaste -** An interface exposing the members that a garbage object should possess. All passed in garbage objects should implement this interface.
2. **IGarbageDisposalStrategy -** An interface exposing the members that a garbage disposal strategy should possess. All passed in garbage disposal strategies should implement this interface.
3. **IGarbageProcessor -** An interface implemented by the framework’s Garbage Processor.
4. **IStrategyCollector -** An interface exposing the members that a strategy collector should implement. Any strategy collector passed to the Garbage Processor should implement this interface.
5. **IProcessingData -** An interface exposing the members that a processing data object should possess. The ProcessGarbage method of a strategy should return an object implementing this interface.

**Concrete classes**

1. **GarbageProcessor -** A class containing the main functionality of the framework. It holds an **IStrategyCollector** object to contain the strategies and exposes a method for **processing waste**, which resolves the disposal strategy needed based on the passed in waste’s disposal attribute.
2. **StrategyCollector -** A default internal implementation to the IStrategyCollector interface, used to provide the default operation of the Waste Disposal Framework. It exposes methods for **adding** a new disposal strategy with a new attribute, **removing** an attribute and its corresponding strategy and **returning a read only copy** of the attribute-strategy collection. The internal collection should hold only **unique** attribute types.

### User Input

1. **ProcessGarbage {name}|{weight}|{volumePerKg}|{type}**
   1. Receives a garbage for processing with the given **weight** and **volume per kilogram**.
   2. The **{name}** will be a non-empty string consisting of only alphabetical characters.
   3. The **{weight}** will be a valid positive double.
   4. The **{volumePerKg}** will be a valid positive double.
   5. The **{type}** will always be one of the following **“Recyclable”, “Burnable”** or **“Storable”**.
   6. In case the garbage is allowed by the management criteria print **“{garbage Weight} kg of {garbageName} successfully processed!”**.
   7. Alternatively, if the garbage is not allowed print **“Processing Denied!”.**
2. **ChangeManagementRequirement {energyBalance}|{capitalBalance}|{garbageType}**
   1. Receives a new management requirment with the given **criteria** and values **energyBalance** and **capitalBalance**, denying the specified **garbageType**.
   2. The **{energyBalance }** will be a valid double.
   3. The **{capitalBalance}** will be a valid double.
   4. The **{garbageType}** will always be one of the following **“Recyclable”, “Burnable”** or **“Storable”**.
   5. When a new managemet requirement is received you should print **“****Management requirement changed!”**.
3. **State**
   1. Prints information about the current balance of the Recycling Station in the following format **“****Energy: {energyBalance} Capital: {capitalBalance}”**.

### Input

* Input will be received through the console.
* Each command will come on a new line.
* The input ends when you receive the command “**TimeToRecycle**”.

### Output

* Print each input command’s resulting message on a new line.

### Constraints

* **All floating point numbers should be printed to the second decimal place** (ex. EnergyBalance, Weight, VolumePerKg… e.t.c.)
* All numbers passed in through the input will be valid doubles.
* All strings passed in through the input will be valid non-empty strings consisting of only alphabetical characters.
* All commands passed through the input will be valid.
* The last command received through the input will always be “**TimeToRecycle**”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ProcessGarbage Glass|10|1.14|Recyclable ProcessGarbage Wood|33.156|3.657|Burnable ProcessGarbage UsedShampooBottles|2.25|1.57|Storable Status ChangeManagementRequirement 500|5000|Storable **ProcessGarbage OldTires|13.88|5.99|Storable Status TimeToRecycle** | **10.00 kg of Glass successfully processed! 33.16 kg of Wood successfully processed! 2.25 kg of UsedShampooBottles successfully processed! Energy: 90.84 Capital: 3997.70 Management requirement changed! Processing Denied! Energy: 90.84 Capital: 3997.70** |
| **Comment** | |
| **We first receive 10kg of Glass with 1.14 cm3 per kg, using the recycling formula we get 400 \* 10 = 4000 capital and use (10 \* 1.14) \* 0.5 = 5.7 energy. We then process the Wood, using the formula for burnable garbage we get (33.156 \* 3.657) \* (1 - 0.2) = 121.2515 \* 0.8 = 97.0012 energy. Accounting for the 5.7 energy used in the recycling of the Glass that leaves us with 97.0012 - 5.7 = 91.3012 energy profit. We then process the Used Shampoo Bottles, using the formula for storable garbage we get => (2.25 \* 1.57) \* 0.65 = 2.2961 used capital and (2.25 \* 1.57) \* 0.13 =  0.4592 used Energy. Making the calculations this leaves us with a total of 4000 - 2.2961 = 3997.7039 Capital and 91.3012 - 0.4592 = 90.8420 Energy which we print to the second decimal place (as mentioned in the constraints) when the Status command is called. The management requirement is then changed to require that the Recycling Station has at least 500 Energy and 5000 Capital or otherwise to deny processing Storable garbage. Since currently both the Capital and the Energy are below the criteria requirements and the next processing request is of type Storable it is denied. We then receive another Status command, since the previous processing request was denied the values are the same as the ones from the previous Status command.** | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **ProcessGarbage Rubber|17.13|2.01|Storable** **ProcessGarbage PlasticBottles|0.866|5.31|Storable ProcessGarbage OldBooks|33.44|0.9157|Recyclable**  **Status ChangeManagementRequirement 200|-500|Recyclable ProcessGarbage WindowGlass|377.13|1.14|Recyclable ProcessGarbage WoodenFurniture|134.93|2.87|Burnable**  **ProcessGarbage PaperBags|1.77|8.18|Recyclable**  **ChangeManagementRequirement 0|20000|Burnable**  **ProcessGarbage WoodLogs|500|10|Burnable**  **Status TimeToRecycle** | **17.13 kg of Rubber successfully processed! 0.87 kg of PlasticBottles successfully processed! 33.44 kg of OldBooks successfully processed!**  **Energy: -20.38 Capital: 13350.63 Management requirement changed! Processing Denied! 134.93 kg of WoodenFurniture successfully processed! 1.77 kg of PaperBags successfully processed! Management requirement changed! Processing Denied! Energy: 282.18 Capital: 14058.63** |

### Tasks

### Refactor and use the Framework

The framework you have is decently written, but it might violate some of the principles of good object oriented design and general coding guidelines or contain a few minor bugs. Refactor any problems you find in the framework. You are allowed to refactor any parts of it, if you think that will improve its overall quality. Your code **MUST** use the provided framework, spend some time to understand how it works and build your code to use the provided functionality.

**20 score**

### Correct results in Judge

The framework provides some functionality, but it doesn’t cover the entire task, implement the rest of the business logic, meeting the specification requirements. Test your code in the automated Judge system to get a real time measurement of the correctness of your program.

**20 score**

### High Quality

Achieve good separation of concerns using abstractions and interfaces to decouple classes, while reusing code through inheritance and polymorphism. Your code should be modular, reusable and extendable, following the best practices of OOP and High Quality Code.

**40 score**

### Unit Testing

Test the **Garbage Processor** and **Strategy Collector** classes provided by the framework. Extensive testing might require you to have some of the core logic implemented, in order to cover all cases. Mock all dependencies when testing a class.

**20 score**