OOP Advanced Exam - Recycling Station

Waste collection and recycling is a big problem for modern societies, as the consumption of goods increases so does the produced waste, thus an 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 and printing its current state (its resources). Your Recycling Station should start with **0 Energy** and **0 Capital.**

Models

The models with which your program should work are the following: **Waste, ProcessingData** and **GarbageDisposalStrategy**.

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

A **Garbage Disposal Strategy** should implement a single metod **processGarbage** that takes in a **garbage** object and processes it. Each garbage type should have a corresponding **disposal strategy** that knows how to process it and produces a result in the form of a **Processing Data** object. The processing results should be as follows:

	Energy Produced	Energy Used	Capital Earned	Capital Used
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

A **Processing Data** object is used to transfer information within the application, it should have the following members - **EnergyBalance** and **CapitalBalance**.

Framework Overview

You will be given a package “wasteDisposal” which functions as your framework - it exposes its own interfaces and classes as needed. Only code that should **belong to the framework** should be written in that package.

The framework receives and maps strategies for disposal of garbage to annotation classes implementing a meta-annotation exposed by the Framework, then when receiving a waste object it searches for a strategy to process it, based on its annotation.

A meta annotation is simply an annotation that can only be put on other annotations. Here's an example:

```
@Target(ElementType.ANNOTATION_TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyMetaAnnotation {
}
```

```
@MyMetaAnnotation
public @interface MyNormalAnnotation {
}
```

The framework has the following classes:

Annotations

1. **@Disposable -** A meta-annotation targeting other annotations, the implementing annotation should be used to map the Garbage Disposal Strategy. The provided framework requires the passed in Waste object to implement at least one annotation which has the Disposable meta-annotation.

Interfaces/Contracts

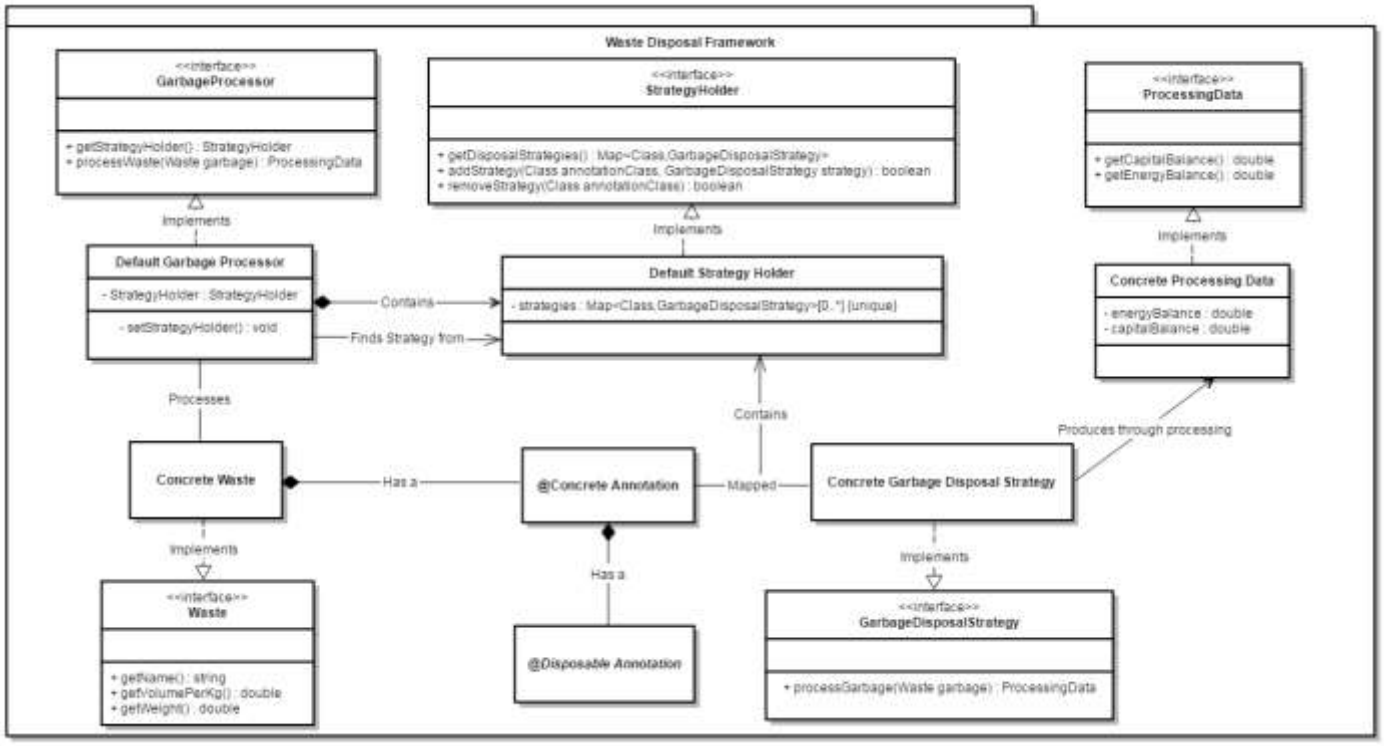
1. **Waste -** An interface exposing the members that a garbage object should possess. All passed in garbage objects should implement this interface.
2. **GarbageDisposalStrategy -** An interface exposing the members that a garbage disposal strategy should possess. Garbage disposal strategies are the objects who understand the actual processing of the garbage and calculate and return the results of the process in the form of a Process Data object.
3. **GarbageProcessor -** An interface implemented by the framework's Garbage Processor.
4. **StrategyHolder -** An interface exposing the members that a Strategy Holder should implement. Any Strategy Holder passed to the Garbage Processor should implement this interface.
5. **ProcessingData -** 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. **DefaultGarbageProcessor -** A default implementation of a Garbage Processor, it holds a collection of **Garbage Disposal Strategies** mapped to specific annotation classes through means of a **Strategy Holder** object. The Garbage Processor also exposes a method **processWaste** which resolves what strategy should be used to process the passed in garbage and then delegates the actual processing to that strategy.
2. **DefaultStrategyHolder -** A default internal implementation to the StrategyHolder interface, used to provide the default operation of the Waste Disposal Framework. It exposes methods for **adding** a new Annotation Class -> Garbage Disposal Strategy mapping, **removing** an Annotation Class and its corresponding strategy and **returning a read only copy** of the Annotation Class->Garbage Disposal Strategy collection. The internal collection should hold only **unique** annotation classes.

UML Diagram

A UML Diagram is provided for you, so you can better understand the structure and workflow of the framework.



User Input

1. **ProcessGarbage {name}|{weight}|{volumePerKg}|{type}**
    a. Receives a garbage for processing with the given **name, weight** and **volume per kilogram** from the specified **type**.
    b. The **{name}** will be a non-empty string consisting of only alphanumerical characters.
    c. The **{weight}** will be a valid positive double.
    d. The **{volumePerKg}** will be a valid positive double.
    e. The **{type}** will always be one of the following **"Recyclable", "Burnable"** or **"Storable"**.
    f. After processing the passed in garbage print the following message **"{garbage Weight} kg of {garbageName} successfully processed!"**.
2. **Status**
    a. 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 alphanumerical 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 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 13.88 kg of OldTires successfully processed! Energy: 80.03 Capital: 3943.66
Comment	
We first receive **10**kg 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 **gain** (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. We receive **13.88**kg of **OldTires** to process, using the formula for Storable garbage we get (13.88 * 5.99) * 0.65 = **54.0418 used Capital** and (13.88 * 5.99) * 0.13 = **10.8084 used Energy.** Calculating with what we got so far we're left with 3997.7039 - 54.0418 = **3943.6621 Capital** and 90.8420 - 10.8084 = **80.0336 Energy.** We then receive another **Status** command, so we print the values (to the second decimal digit) **Energy: 80.03** and **Capital: 3943.66**	

Input	Output
ProcessGarbage TreeShavings|15.32|0.254|Burnable **ProcessGarbage ShatteredGlass|3.77|850.15|Recyclable** **Status** **ProcessGarbage OldTires|18.88|6.99|Storable** **ProcessGarbage CardPyramid|55.1387|3.31|Burnable** **Status** **TimeToRecycle**	**15.32 kg of TreeShavings successfully processed!** **3.77 kg of ShatteredGlass successfully processed!** **Energy: -1599.42 Capital: 1508.00** **18.88 kg of OldTires successfully processed!** **55.14 kg of CardPyramid successfully processed!** **Energy: -1470.57 Capital: 1422.22**

Tips – Really recommend reading these

Getting the class of an annotation is done through the .annotationType() method. Trying to get it through .getClass() method will return a proxy that carries no information about the annotation class and will be of little use to you.

Java doesn't offer any inbuilt functionality to dynamically convert a primitive class to its wrapper class, however this functionality can easily be achieved with the use of a Map, mapping the primitive class to its wrapper class. All wrapper classes for primitives also have a constructor that takes in a string.

Tasks

1. 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, **except the interfaces (which it still has to implement)**, 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

2. 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.

NOTE: Competitive tests 11-15 and zero test 3 are reserved for the Bonus Task. Passing Tests 1-10 will grant you the full 20 score for this task.

20 score

3. High Quality

Achieve good separation of concerns using abstractions and interfaces to decouple classes, while reusing code through inheritance and polymorphism. Your classes should have strong cohesion - have single responsibility and loose coupling - know about as few other classes as possible.

25 score

4. Reusability

Your code should be modular, reusable and extendable, following the best practices of OOP and High Quality Code. Extend your program to be able to handle **ANY** garbage type - in other words newly introduced model classes should be able to work with your program without the need to rewrite any core logic.

15 score

5. Unit Testing

Test the framework classes implementing the **Garbage Processor** and **Strategy Holder** interfaces (by default those should be **DefaultGarbageProcessor** and **DefaultStorageHolder**). 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

6. *Bonus Task - Extended Functionality

Extend your program to support a new command **ChangeManagmentRequirement.**

User Input

3. **ChangeManagementRequirement {energyBalance}|{capitalBalance}|{garbageType}**
    a. Receives a new management requirment with the given **energyBalance** and **capitalBalance**, denying the specified **garbageType**.
    b. The **{energyBalance }** will be a valid double.
    c. The **{capitalBalance}** will be a valid double.
    d. The **{garbageType}** will always be one of the following **"Recyclable", "Burnable"** or **"Storable"**.
    e. When a new managemet requirement is received you should print **"Management requirement changed!"**.

Your program should be able to deny processing a specific type of garbage when **either** the Recycling Station's **Energy OR Capital** is **under** the requirment specified ammounts. If a ProcessingGarbage command is denied, instead of it's usual message it prints **"Processing Denied!".**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 regardless of Energy or Capital values).

For an example of how the management requirements work - if the current management requirment is set to 100 Energy and 150 Capital, with denied garbage Type - Recyclable, and the Recycling Station has 90 Energy and 200 Capital. Receiving an input to Process a garbage of type Recyclable, the processing request will be **DENIED**, because currently the Recycling Station's Energy (90) is bellow the requirment criteria for energy (100).

It is important to note that when a management requirment is received, it stays in effect untill a new management requirement overwrites it. Management requirements also impose their check **ONLY** on the specified garbage type – if in the above example the garbage type we had received was of any other type than Recyclable it would have been processed regardless of the Recycling Station's Energy and Capital values.Check the examples to better grasp how the management requirement works.

NOTE: Competitive tests 11-15 and zero test 3 are reserved for this Task. Passing Tests 11-15 will grant you the 10 score for this task.

Bonus 10 score

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|3000|Storable ProcessGarbage OldTires|13.88|5.99|Storable ProcessGarbage CrumbledPaper|3.68|0.45|Recyclable 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! 3.68 kg of CrumbledPaper successfully processed! Energy: 90.01 Capital: 5469.70
Comment	

We first receive **10**kg 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 **gain** (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 3000 Capital** or otherwise to deny processing **Storable** garbage. Since currently the Recycling Station's **Energy** is **below the criteria** requirements and the next processing request is of type Storable it is **denied**. Afterwards we receive **3.68**kg of **CrumbledPaper** to recycle, the processing station still has **90.8420 Energy**, which is still under the Management Requirement, but since the garbage type is **Recyclable**, the Management Requirement **doesn't apply to it**. We process the **CrumbledPaper** and we get 3.68 * 400 = **1472 Capital** and (3.68 * 0.45) * 0.5 = **0.8280 Energy used**. Calculating to our previous totals that leaves us with 3997.7039 + 1472 = **5469.7039 Capital** and 90.8420 – 0.8280 = **90.0140 Energy**. We then receive another **Status** command and we print the new values – **Energy: 90.01** and **Capital: 5469.70**.