# OOP Advanced Exam – Emergency

The “Pegboard Nerds” PN Organization and the “Nothing but Thieves” NBT Corporation have established the PRRM Services, in order to fight emergencies all over the world. PRRM goes for Prepare, Respond, Recover and Mitigate – the core phases in emergency management.

### Overview

The PRRM Services have a specified Emergency Management System (EMS) which processes emergencies in order of their level and registration time. There are specified Emergency Managing Centers for each type of emergency.

The emergencies have a different level and a registration time. They are processed in order of entrance. The emergencies are being queued on the EMS, upon registration. They can then be processed, one by one, and passed to the respective emergency center.

The registered centers of the EMS are also queued. When an emergency is processed, it is passed to the first registered center, of the same type of course. Then, the emergency center is passed to the end of the current center queue, and the next center in order becomes the first, to which the next emergency will be passed. This is done, because the centers need time to respond to the given emergency, before they can take another one.

The emergency centers also have an exact amount of emergencies they can process, before they retire. When an emergency center retires, due to processing its last emergency, it simply disappears from the center queue.

The EMS should also provide live statistics, about the emergencies from the emergency centers, to the PRRM Services’ founders – The “Pegboard Nerds” and the “Nothing but Thieves”, so that they can invest the needed money, in order to fix the damage, save the citizen, and fix the public order.

### Models

There are 3 types of Emergencies:

* **Public Property Emergency** – represents fires, earthquakes, car crashes and other emergencies of this type.
  + Has a description, which is a string.
  + Has an emergency level, which can be “Minor”, “Major”, “Disaster”.
  + Has a registration time.
  + Has property damage, which is an integer.
* **Public Health Emergency** – represents human or animal injuries and other emergencies of this type.
  + Has a description, which is a string.
  + Has an emergency level, which can be “Minor”, “Major”, “Disaster”.
  + Has a registration time.
  + Has number of casualties, which is an integer.
* **Public Order Emergency** – represents crimes, riots, public disorder and other emergencies of this type.
  + Has a description, which is a string.
  + Has an emergency level, which can be “Minor”, “Major”, “Disaster”.
  + Has a registration time.
  + Has a status – can be a special case, or not.

There are also 3 types of Emergency Managing Centers:

* **Fireman Service Center** – manages the Public Property Emergencies.
  + Has a name, which is a string.
  + Has an amount of emergencies it can process, which is an integer.
  + Holds all of the emergencies it processed.
* **Medical Service Center** – manages the Public Health Emergencies.
  + Has a name, which is a string.
  + Has an amount of emergencies it can process, which is an integer.
  + Holds all of the emergencies it processed.
* **Police Service Center** – manages the Public Order Emergencies.
  + Has a name, which is a string.
  + Has an amount of emergencies it can process, which is an integer.
  + Holds all of the emergencies it processed.

### Functionality

The core logic of the PRRM Services is interpreted through several commands.

* **RegisterPropertyEmergency|{description}|{level}|{registrationTime}|{propertyDamage}**
  + Registers a Public Property Emergency, with the **given description**, **level**, **registration time** and **property damage**.
  + Upon execution, the command should return the following result:
  + **“Registered Public Property Emergency of level {level} at {registrationTime}.”**
  + The level and the registration time are the ones given from the input.
* **RegisterHealthEmergency|{description}|{level}|{registrationTime}|{casualties}**
  + Registers a Public Health Emergency, with the **given description**, **level**, **registration time** and **amount of casualties**.
  + Upon execution, the command should return the following result:
  + **“Registered Public Health Emergency of level {level} at {registrationTime}.”**
  + The level and the registration time are the ones given from the input.
* **RegisterOrderEmergency|{description}|{level}|{registrationTime}|{status}**
  + Registers a Public Order Emergency, with the **given description**, **level**, **registration time** and **case status**.
  + Upon execution, the command should return the following result:
  + **“Registered Public Order Emergency of level {level} at {registrationTime}.”**
  + The level and the registration time are the ones given from the input.
* **RegisterFireServiceCenter|{name}|{amountOfEmergencies}**
* **RegisterMedicalServiceCenter|{name}|{amountOfEmergencies}**
* **RegisterPoliceServiceCenter|{name}|{amountOfEmergencies}**
  + Registers an emergency center with **the given name**, and **amount of emergencies** it can process.
  + Upon execution, the command should return one of the following result, depending on the command:
  + **“Registered Fire Service Emergency center – {name}.”**
  + **“Registered Medical Service Emergency center – {name}.”**
  + **“Registered Police Service Emergency center – {name}.”**
  + The name is the one given from the input.
* **ProcessEmergencies|{type}**
  + Processes all registered emergencies, of the given type, one by one, in the order of their entrance.
  + The type can be “Property”, “Health”, “Order”.
  + Redistributes each emergency to the next center in order.
  + Upon execution, the command should return one of the following results:
  + **“Successfully responded to all {type} emergencies.”** – if it has managed to process all the registered emergencies from the **given type**.
  + **“{type} Emergencies left to process: {amountOfEmergenciesLeft}.”** – if it has not managed to process all the registered emergencies from the **given type**. The main reason for that to happen might be the lack of centers, due to their retirement.
* **EmergencyReport**
  + Brings up live statistics about all the emergency situation.
  + The following should be returned as result of the execution:

**“PRRM Services Live Statistics**

**Fire Service Centers: {amountOfFireServiceCenters}**

**Medical Service Centers: {amountOfMedicalServiceCenters}**

**Police Service Centers: {amountOfPoliceServiceCenters}**

**Total Processed Emergencies: {amountOfAllProcessedEmergencies}**

**Currently Registered Emergencies: {amountOfCurrentlyRegisteredEmergencies}**

**Total Property Damage Fixed: {amountOfTotalPropertyDamageOfAllProcessedEmergencies}**

**Total Health Casualties Saved: {amountOfHealthCasualtiesOfAllProcessedEmergencies}**

**Total Special Cases Processed: {amountOfSpecialCasesProcessedFromAllProcessedEmergencies}**

* + You need to print all of the **currently functioning** emergency centers’ count. Do **NOT** count the **retired ones**.
  + After that you need to print the amount of all processed emergencies, and the amount of currently registered emergencies, awaiting for processing.
  + Then you have to print the total property damage fixed, health casualties saved, and special cases processed, which is a statistic you must get from the processed emergencies.
* **EmergencyBreak**
  + The input-terminating command. When entered – ends the input sequence.

### Input

* The input will come in the form of commands in the exact format specified above.
* The input sequence ends, when you receive the command **“EmergencyBreak”**.

### Output

* The only output you should print is the one from the result of executing commands.

### Packages

* annotations
* collection
* commands
* core
* engines
* enums
* interpreters
* io
* models
  + centers
  + emergencies
* utils

### Classes

* InjectArgs
* InjectType
* EmergencyRegister
* Register
* BaseCommand
* EmergencyReportCommand
* Executable
* ProcessEmergenciesCommand
* RegisterFireServiceCenterCommand
* RegisterHealthEmergencyCommand
* RegisterMedicalServiceCenterCommand
* RegisterOrderEmergencyCommand
* RegisterPoliceServiceCenterCommand
* RegisterPropertyEmergencyCommand
* EmergencyManagementSystem
* ManagementSystem
* Engine
* EmergencyLevel
* Status
* CommandInterpreter
* Interpreter
* ConsoleReader
* ConsoleWriter
* Reader
* Writer
* BaseEmergencyCenter
* EmergencyCenter
* FiremanServiceCenter
* MedicalServiceCenter
* PoliceServiceCenter
* BaseEmergency
* Emergency
* HealthEmergency
* OrderEmergency
* PropertyEmergency
* RegistrationTime
* RegistrationTimeImpl

### Constraints

* You can safely assume that there will be NO invalid input.
* The input names of the emergency centers and the descriptions of the emergencies will be strings, which can consist of any ASCII character.
* The input levels of the emergencies will either be “Minor”, “Major” or “Disaster”.
* The registration time will always be valid.
* The input property damage and health casualties will be valid integers in range [0, 231 – 1].
* The input case status will either be “Special” or “Non-Special”.
* The input amount of emergencies to process, for the centers, will be a valid integer in range [0, 500].
* Allowed time/memory: 250ms/16MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **RegisterPropertyEmergency|Test|Minor|12:24 25/02/2016|2500**  **RegisterHealthEmergency|SomeTest|Major|12:24 26/02/2016|10**  **RegisterOrderEmergency|SomeTest1|Disaster|12:24 27/02/2016|Special**  **RegisterFireServiceCenter|FireService|2**  **RegisterMedicalServiceCenter|Pirogov|1**  **RegisterPoliceServiceCenter|3TORPU|1**  **ProcessEmergencies|Property**  **EmergencyReport**  **EmergencyBreak** | **Registered Public Property Emergency of level Minor at 12:24 25/02/2016.**  **Registered Public Health Emergency of level Major at 12:24 26/02/2016.**  **Registered Public Order Emergency of level Disaster at 12:24 27/02/2016.**  **Registered Fire Service Emergency Center – FireService.**  **Registered Medical Service Emergency Center – Pirogov.**  **Registered Police Service Emergency Center - 3TORPU.**  **Successfully responded to all Property emergencies.**  **PRRM Services Live Statistics**  **Fire Service Centers: 1**  **Medical Service Centers: 1**  **Police Service Centers: 1**  **Total Processed Emergencies: 1**  **Currently Registered Emergencies: 2**  **Total Property Damage Fixed: 2500**  **Total Health Casualties Saved: 0**  **Total Special Cases Processed: 0** |

### Tasks

### Task 1: Refactor and use the given code

You will be given a class which represents a custom collection. That class is used as a holder for the emergencies, because it can process them in order of their entrance. You will need to fix it so that it can work with all emergencies. You will also be given a class which represents the EMS.

You are permitted to modify everything as long as it follows the application logic. However you MUST use the given code.

You will also be given a class which represents the registration time. It is completely implemented.

The use of the given code in combination with your code is an absolute necessity. Otherwise there might be bugs.

Refactor the given code, if necessary, so that it follows the best practices from the OOP Advanced course.

**20 score**

### Task 2: Bug fixing

It is not necessary for it to be called a bug... Rather than that, it can be called an unsuitable implementation. See if you can fix the **EmergencyRegister** class so that it can work well with all types of emergencies.

**10 score**

### Task 3: Correct results in Judge

Combine the given code with your code, in order to implement the needed functionality for the application to work properly. You will be given 8 judge tests in total – 3 zero tests and 5 competitive. The 3-rd zero test will be optional.

**20 score**

### Task 4: 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.

**20 score**

### Task 5: Reflection usage

You need to use Reflection at least once in your code, and it needs to be connected to the main logic, for it to be considered valid.

**10 score**

### Task 6: Unit Testing

Implement unit tests for the **RegistrationTime** class and the **EMS** class. Make sure you test everything completely and correctly.

**30 score**