ENERGETIC SYSTEM SIMULATOR

Duțu Alin Călin

CONTENTS

1 INTRODUCTION

1.1 Context

The electricity network of a country is a complex system in which the government must supervise the quality and stability of the system and regulate it in order to be affordable for people and sustainable in the long term.

1.2 Objectives

This project aims to provide an advanced overview of Object-Oriented programming by developing a platform for administering a country's electrical network using advanced OOP concepts.

The following objectives will be covered after implementing the program:

- Developing basic abilities of organization and OOP design
- Developing the ability to write generic code
- Understanding how and when to use design patterns
- Adopting a specific coding design and style for Object Oriented Programming

2 PROPOSED SOLUTION

2.1 Solution presentation

The project aims to implement an energetic system simulator in which the considered entities will have the main purpose of remaining in business and avoiding bankruptcy.

There 3 entity categories:

- Producers - The entity which is considered to produce and sell energy
- Consumers - The entity that buys and uses the produced energy
- Distributors - The entity which acts as an intermediary between the Producers and the Consumers

These entities will try to take action and finish their specific responsibilities over a specific number of rounds simulating the passing months.

Each round comes with new updates for the simulator that will be applied at the start of each round. On a single round, the following actions are executed in order:

1. If it is the first round, the distributors will choose the producers based on their choosing strategy
2. If it is not the first round
   (a) Checks if all distributors are bankrupt, in which case the simulation is finished
   (b) Adds new consumers and updates costs for distributors based on the round instructions
   (c) The consumers will be checked for bankruptcy, in which case the corresponding actions will be taken
3. Distributor price list is updated
4. The consumers receive their monthly salary
5. The distributors will be checked for bankruptcy, in which case the distributor is eliminated and its contracts closed
6. Assigns new contracts for consumers that are not bankrupt and have no contract with any distributor
7. The consumers pay their bills and the distributors will have their budgets updated
8. The consumers will be checked for bankruptcy, in which case the corresponding actions will be taken
9. If it is not the first round

(a) Producer's debit per distributor is updated based on the round instructions
(b) The distributors who seek new producers will set new contracts based on their choosing strategy
(c) Statistics of distributors which have contracts with producers are extracted

2.2 Round flow

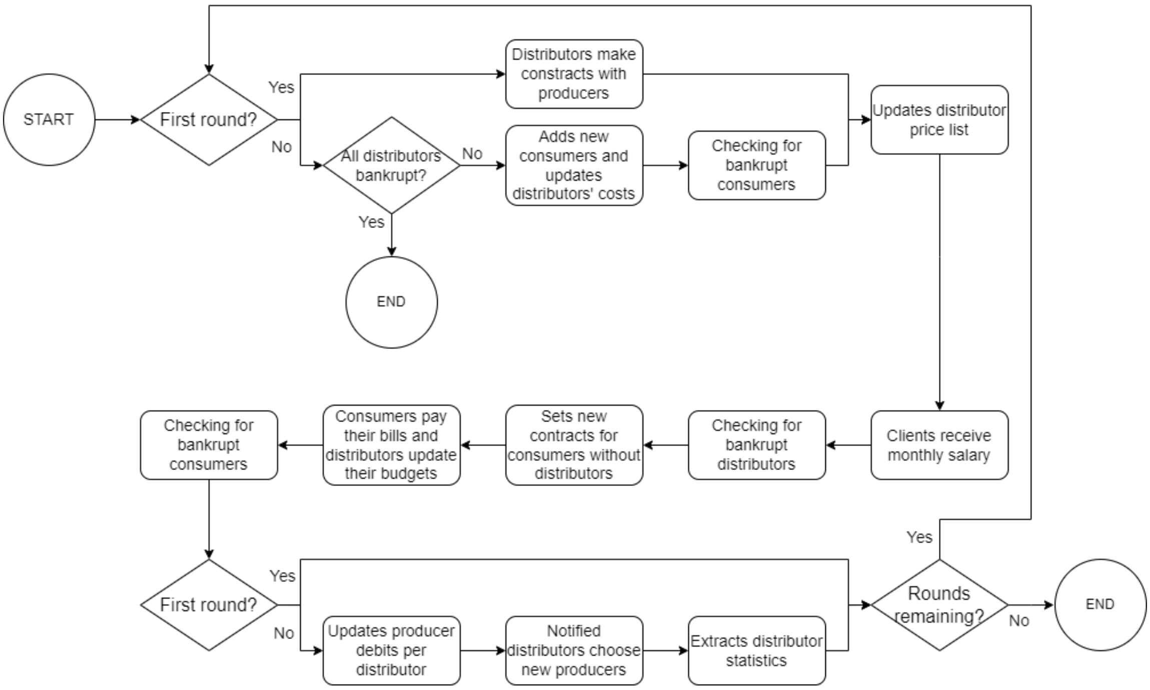

Figure 1: Round flow

3 IMPLEMENTATION DETAILS

3.1 Proposed implementation

The simulator starts its execution by reading data from a given input file that has data about multiple producers, distributors, and consumers that are going to be registered. After initialization, the first round will begin and it will respect the presented rounds flow [Fig 1]

3.1.1 Producers

Description

The producers are the entities that produce energy and deliver it to the distributors. Energy can be produced in many ways thus producers can be categorized by the energy type into producers that produce renewable energy (wind, solar, hydro) and non-renewable energy (coal, nuclear).

Additionally, the producers will have an id for identification, a price per kWh, the monthly quantity of energy it can offer to one distributor, and the maximum number of distributors it can set a contract with.

JSON representation

```
 {
    "id": 0,
    "energyType": "WIND",
    "maxDistributors": 10,
    "priceKW": 0.01,
    "energyPerDistributor": 2695
}
```

Actions

The producers will meet their obligations to offer the requested energy to distributors based on the contracts it has. However, producers can change their energy price for a month based on the indications provided by the simulator and will notify the associated distributors about the event, a mechanic implemented with the Observer Design Pattern.

3.1.2 Distributors

Description

The distributors act as intermediaries between producers and consumers. One distributor can sign contracts with multiple producers to fulfill its necessary energy debit. The distributor will opt for one of the 3 strategies:

- Green Strategy - The distributor prioritizes producers with renewable energy at first, the lowest prices secondly, and the quantity offered at last
- Price Strategy - The distributor prioritizes producers by the price first, and the quantity offered last
- Quantity Strategy - The distributor prioritizes the producers by the quantity offered

Distributors can also make contracts with consumers when requested and get paid at the stated price. If a consumer is unable to pay the bill, the distributor will add a penalty to the next month's bill. If the consumer is unable to pay the next bill, the contract will be terminated.

In addition, a distributor has an ID for identification, the number of months until the contract with the chosen producer is due, an initial budget, an initial infrastructure cost, the amount of energy needed, and the strategy for selecting producers.

JSON representation

```json
{
    "id": 0,
    "contractLength": 6,
    "initialBudget": 60,
    "initialInfrastructureCost": 24,
    "energyNeededKW": 1930,
    "producerStrategy": "GREEN"
}
```

Actions

The strategies chosen by the distributors are implemented using the Strategy Design Pattern. Based on the provided input, the pattern will sort the producers accordingly and then let distributors set contracts with the best producers for their needs.

Additionally, the distributors, as soon as they are notified by the producer of a price change, can opt for choosing other producers. The listening mechanic for distributors is implemented using the Observer Design Pattern.

Moreover, distributors have a cost of infrastructure which reflects the cost of receiving, maintaining, and distributing energy. This cost is monthly and it is updated by the system.

In relationship with consumers, distributors will set a penalty, in case they can't the bill to the distributor, in addition to the next month's bill according to the following formula:

$$Math.round(Math.floor(1.2 * old_bill)) + new_bill$$

3.1.3 Consumers

Description

The consumer is the entity that buys energy from distributors for its benefit.

JSON representation

```
{
    "id": 0,
    "initialBudget": 150,
    "monthlyIncome": 28
}
```

Actions

The consumers, besides receiving salary and paying bills to the subscribed distributors can move to other distributors, but only when their contract with their former distributor has expired.

In case a consumer opts for a new distributor, it has to pay its penalties to its old distributor, in addition to the new bill. If it is unable to do so, it will be considered bankrupt.

3.1.4 Input - Output

The input is the information provided by the system. In the first round, the input provided will load all the entities with their corresponding data. In the next rounds, the following changes can take place based on the system's input:

- Monthly price changes for producers
- Monthly changes in infrastructure costs for distributors
- New clients added

One such example can be found below:


```
{
  "numberOfTurns": 6,
  "initialData": {
    "consumers": [
      {
        "id": 0,
        "initialBudget": 150,
        "monthlyIncome": 28
      }
    ],
    "distributors": [
      {
        "id": 0,
        "contractLength": 6,
        "initialBudget": 60,
        "initialInfrastructureCost": 24,
        "energyNeededKW": 1930,
        "producerStrategy": "GREEN"
      }
    ],
    "producers": [
      {
        "id": 0,
        "energyType": "WIND",
        "maxDistributors": 10,
        "priceKW": 0.01,
        "energyPerDistributor": 2695
      }
    ]
  },
  "monthlyUpdates": [
    {
      "newConsumers": [
        {
```

```
      "id": 0,
      "initialBudget": 140,
      "monthlyIncome": 24
    }
  ],
  "distributorChanges": [
    {
      "id": 0,
      "infrastructureCost": 20
    }
  ],
  "producerChanges": [
    {
      "id": 0,
      "energyPerDistributor": 6319
    }
  ]
},
{
  "newConsumers": [
    {
      "id": 0,
      "initialBudget": 56,
      "monthlyIncome": 97
    }
  ],
  "distributorChanges": [
    {
      "id": 0,
      "infrastructureCost": 25
    }
  ],
  "producerChanges": [
    {
      "id": 0,
      "energyPerDistributor": 4430
    }
  ]
},
{
  "newConsumers": [
    {
```

```
        "id": 0,
        "initialBudget": 188,
        "monthlyIncome": 43
      }
    ],
    "distributorChanges": [
      {
        "id": 0,
        "infrastructureCost": 17
      }
    ],
    "producerChanges": [
      {
        "id": 0,
        "energyPerDistributor": 9965
      }
    ]
  },
  {
    "newConsumers": [
      {
        "id": 0,
        "initialBudget": 55,
        "monthlyIncome": 65
      }
    ],
    "distributorChanges": [
      {
        "id": 0,
        "infrastructureCost": 17
      }
    ],
    "producerChanges": [
      {
        "id": 0,
        "energyPerDistributor": 2569
      }
    ]
  },
  {
    "newConsumers": [
      {
```

```
        "id": 0,
        "initialBudget": 163,
        "monthlyIncome": 27
      }
    ],
    "distributorChanges": [
      {
        "id": 0,
        "infrastructureCost": 13
      }
    ],
    "producerChanges": [
      {
        "id": 0,
        "energyPerDistributor": 9077
      }
    ]
  },
  {
    "newConsumers": [
      {
        "id": 0,
        "initialBudget": 140,
        "monthlyIncome": 85
      }
    ],
    "distributorChanges": [
      {
        "id": 0,
        "infrastructureCost": 16
      }
    ],
    "producerChanges": [
      {
        "id": 0,
        "energyPerDistributor": 4523
      }
    ]
  }
 ]
}
```

The output of each round will have evidence of all entities and some additional statistics. One such example can be found below:


```
{
  "consumers" : [ {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 178
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 152
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 446
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 324
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 208
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 209
  }, {
    "id" : 0,
    "isBankrupt" : false,
    "budget" : 213
  } ],
  "distributors" : [ {
    "id" : 0,
    "energyNeededKW" : 1930,
    "contractCost" : 12,
    "budget" : 261,
    "producerStrategy" : "GREEN",
    "isBankrupt" : false,
    "contracts" : [ {
      "consumerId" : 0,
```

```
      "price" : 22,
      "remainedContractMonths" : 0
    }, {
      "consumerId" : 0,
      "price" : 19,
      "remainedContractMonths" : 1
    }, {
      "consumerId" : 0,
      "price" : 9,
      "remainedContractMonths" : 2
    }, {
      "consumerId" : 0,
      "price" : 14,
      "remainedContractMonths" : 3
    }, {
      "consumerId" : 0,
      "price" : 4,
      "remainedContractMonths" : 4
    }, {
      "consumerId" : 0,
      "price" : 12,
      "remainedContractMonths" : 5
    }, {
      "consumerId" : 0,
      "price" : 12,
      "remainedContractMonths" : 5
    } ]
  } ],
  "energyProducers" : [ {
    "id" : 0,
    "maxDistributors" : 10,
    "priceKW" : 0.01,
    "energyType" : "WIND",
    "energyPerDistributor" : 4523,
    "monthlyStats" : [ {
      "month" : 1,
      "distributorsIds" : [ 0 ]
    }, {
      "month" : 2,
      "distributorsIds" : [ 0 ]
    }, {
      "month" : 3,
```

```
      "distributorsIds" : [ 0 ]
    }, {
      "month" : 4,
      "distributorsIds" : [ 0 ]
    }, {
      "month" : 5,
      "distributorsIds" : [ 0 ]
    }, {
      "month" : 6,
      "distributorsIds" : [ 0 ]
    } ]
  } ]
}
```


The input and output entities are created using the Factory Design Patterns since there is a common way of how these classes work. The input gathers its data from an input file using the Jackson library which parses the provided data into JSON readable data. The output gathers the results from the simulation, represented as JSON, converts it to the desired output format using the same Jackson library, and puts it in the output files.

3.2 Testing

For testing the proposed solution, the following methods have been used:

- Checker - Used for testing general cases
- Manual testing - Used for testing corner cases

4 CONCLUSIONS

In conclusion, this program aims to provide an advanced overview of Object-Oriented programming by developing a platform for administering a country's electrical network. The app offers a great and practical example of using advanced techniques such as design patterns as well as attempting to design code with the SOLID principles in mind making it easier to add more functionalities. This program provides a simulator for a country's electrical network that will keep the evidence of multiple consumers, distributors, and producers along with additional stats that will facilitate the administration of such a complex system.

BIBLIOGRAPHY

[1] POO team @ Polytechnic University of Bucharest. Proiect - Etapa 1 - Sistem energetic. `https://ocw.cs.pub.ro/courses/poo-ca-cd/arhiva/teme/2020/proiect/etapa1`. The homework is written in Romanian; Latest access: 24.08.2024;.

[2] POO team @ Polytechnic University of Bucharest. Proiect - Etapa 2 - Sistem energetic. `https://ocw.cs.pub.ro/courses/poo-ca-cd/arhiva/teme/2020/proiect/etapa2`. The homework is written in Romanian; Latest access: 24.08.2024;.