LICENCIATURA EM ENGENHARIA INFORMÁTICA

Programação Orientada a Objetos

Trabalho Prático 2023/2024

Meta 1

Tomás Alexandre Dias Laranjeira

2021135060

Pedro Romão Raimundo Vaz de Paiva

2021134625

Sumário

1. Introdução

O objetivo deste trabalho prático é o desenvolvimento de um simulador de uma habitação que é controlada por vários componentes de domótica interligados entre si. A habitação está dividida em zonas, que por sua vez têm propriedades tais como luz, humidade, temperatura, etc., estas propriedades são inspecionadas por sensores que por sua vez fornecem os valores lidos a regras que são geridas por processadores de regras. Estes decidem o que necessita ser feito de acordo com as leituras dos sensores, e para isso enviam comandos para aparelhos que vão afetar as propriedades da zona em que se encontram. O simulador é controlado por comandos introduzidos pelo utilizador e inclui a noção de passagem de tempo. Esta passagem de tempo é dada por instantes e é controlada através de comandos introduzidos pelo utilizador, sendo completamente independente da passagem do tempo real.

Este trabalho prático deve ser desenvolvido ao longo de duas metas, sendo o relatório presente designado à primeira meta. Esta meta tem como requisitos a leitura e validação de comandos, a leitura do ficheiro de comandos, a construção parcial da interface com o utilizador a construção de classes para cada tipo genérico de componentes e de uma propriedade, a contrução da classe para as zonas, a construção de classes para o processador e para uma regra genérica e a devida organização do projeto em ficheiros .h e .cpp.

2. Opções tomadas

Para a a leitura e verificação de comandos foram criadas na classe interface as seguintes funções:

processaComandos()- esta função verifica se o comando recebido foi o comando sair, se assim for o programa é encerrado, caso contrário chama a função executaComandos enviando como parâmetro o comando recebido.

executaComandos(const string& comando)- esta função compara o comando recebido como parâmetro aos comandos aceites pelo programa, se o comando não for um dos aceites pelo programa, é impressa na janela dos comandos uma mensagem a avisar que o comando é desconhecido e é chamada a função processaComandos. Caso o comando seja algum dos aceites pelo programa são verificados os parâmetros, no caso dos parâmetros não estarem bem introduzidos, é chamada a função sintaxe que envia como parâmetro uma string com o respetivo comando e parâmetros corretos. No caso do comando e parâmetros serem bem introduzidos é chamada a função processa e é impressa na janela de informações uma mensagem de sucesso, após isso é chamada a função processaComandos caso o valor da variável fichAberto seja false, caso contrário é retornada a função.

sintaxe(const string &s)- esta função imprime na janela dos comandos a forma correta de usar o comando em questão e depois chama a função processaComandos caso o valor da variável fichAberto seja false, caso contrário é retornada a função.

sintaxe(const string &s, const string &limite)- esta função imprime na janela dos comandos a forma correta de usar o comando em questão e os valores aceites como parâmetros, depois chama a função processaComandos caso o valor da variável fichAberto seja false, caso contrário é retornada a função.

processa()- esta função imprime na janela dos comandos a mensagem "A processar . . ." por 1 segundo e depois limpa a janela.

carregaComandos(const string &nomeFich)- esta função é chamada quando o comando é introduzido é exec e o respetico nome do ficheiro a ser lido. A função abre o ficheiro e entra num loop em que lê cada linha, enquanto no loop a variável fichAberta é passada a true. Dentro do loop é verificado se o comando recebido foi o comando sair, se assim for o programa é encerrado, caso contrário chama a função executaComandos enviando como parâmetro o comando recebido. No fim do loop a variável fichAberta volta a passar a false e é chamada a função processaComandos().

3. Estruturas usadas

```
class Interface{
private:
    Terminal &terminal;
    Window wComandos;
    Window wInfo;
    Window wHabitacao;
    int iInfo = 1; //incrementa o y do moveto wInfo
    bool fichAberto = false;
public:
    Interface(Terminal &t);
    //Comandos
    void processaComandos();
    void executaComandos(const string& comando);
    void carregaComandos(const string& nomeFich);
    void sintaxe(const string& s);
    void sintaxe (const string& s, const string& limite);
    void processa();
};
```

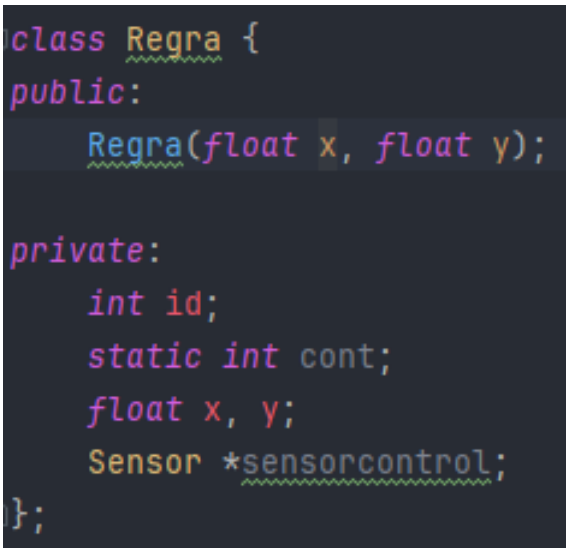
Estrutura da classe Interface

```
class Zona {
private:
    vector<Sensor*> sensores;
    vector<ProcessadorRegras*> processadores;
    vector<Aparelho*> aparelhos;
    int id;
public:
    Zona(int id);

    int getID() const;
};
```

Estrutura da classe Zona

```
class Propriedade {
public:
    Propriedade(string nome, string unidade,int min, int max);
    string getNome() const;
    string getUnidade() const;
    int getMin() const;
    int getMax() const;


private:
    string nome;
    string unidade;
    int min;
    int max;
};
```

Estrutura da classe Propriedade



Estrutura da classe Regra

4. Estruturação do trabalho

O trabalho está organizado em duas pastas.

A pasta lib que contém a classe terminal e o curses.h fornecidos pelo professor.

E a pasta src que contém o main.cpp e várias pastas para as diferentes classes, sendo elas a pasta Aparelhos com a classe Aparelho, a pasta Habitacoes com a classe Habitacao, a pasta Interface com a classe Interface, a pasta Processadores com as classes ProcessadorRegras e a classe Regra, a pasta Propriedades com a classe Propriedade, a pasta Zonas com a classe Zona e por fim a pasta Sensores com a classe Sensor, FumoSensor, HumidadeSensor, LuminosidadeSensor, MovimentoSensor, RadiacaoSensor, SomSensor e TemperaturaSensor.