INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

INSTITUTO POLITÉCNICO DE COIMBRA

Trabalho Prático
Programação Orientada a Objetos

João Duarte Nº2020122715
André Dias Nº2021140917

COIMBRA

1 de janeiro de 2024

Índice

Introdução

O presente trabalho foi realizado no âmbito da cadeira programação orientada a objetos e tem como objetivo a construção de um simulador de uma habitação controlada por componentes de demótica interligados entre si. A linguagem utilizada foi C++.
O simulador é composto por zonas de habitação, que têm propriedades que são inspecionadas por sensores, os quais fornecem os valores lidos a regras que são geridas por processadores de regras que determinam o que fazer em função das leituras dos sensores. Nas zonas existem também aparelhos, os quais afetam as propriedades da zona onde se encontram e reagem a comandos emitidos pelos processadores de regras. O simulador é controlado por comandos escritos pelo utilizador. Existem uns quantos comandos, dando a falsa ideia de dimensão/complexidade, mas na verdade o simulador pouco mais é que um conjunto de objetos interligados entre si e uma interface de utilizador.

Classes

Habitação.h

Engloba a lógica do simulador, fornece a ligação entre comandos e zona dado que tudo tem de passar pela habitação pois é ela quem conhece as zonas. Fornece uma interface completa para interação com o sistema, facilitando a implementação e execução do simulador.

```
private:
    vector<vector<Zona*>> zonas;
    int nLinhas;
    int nColunas;
    int contaInstantes;

    map<string, Processador *> saveProc;
```

```
public:
    Habitacao(int linha, int coluna, map<string, Processador*> &saveProc);

    int getnLinhas() const;
    int getnColunas() const;

    void adicionaZona(int linha, int coluna);
    void removeZona(int Id);
    string listarZonas();

    string modificaPropriedade(int Id, string nomePropriedade, float novoValor);
    string listarPropriedades(int i);

    vector<vector<Zona*>> getZonas();

    string adicionaComponente(int id, char tipoComponente, string parametro);
    string listarComponentes(int id);
    string removeComponente(int id, char tipoComponente, int idComponente);

    string novaRegra(int idZona, int idProcessador, string tipoRegra, int idSensor, float parametro);
    string novaRegra(int idZona, int idProcessador, string tipoRegra, int idSensor, float parametro,float parametro2);
    string listaRegras(int idZona, int idProcRegra);
    string removeRegra(int idZona, int idProcRegra, int idRegra);

    string mudaComandoProcessador(int idZona, int idProcRegra, const string& novoComando);

    string avancaInstante();

    string enviaComandoAparelho(int idZona, int idAparelho, string comando);
    string associaProcessadorAparelho(int idZona, int idProcRegra, int idAparelho);
    string desassociaProcessadorAparelho(int idZona, int idProcRegra, int idAparelho);

    ~Habitacao();

    string guardaEstadoProcessador(int idZona, int idProcRegra, string nome);
    string listaProcessadoresMemoria();
    string carregaEstadoProcessador(string nome);
    string apagaProcessadorMemoria(string nome);

    void resetId();
};
```

Construtor Habitacao:

Parâmetros: linha e coluna para determinar as dimensões da habitação, e saveProc para a inicialização do map de processadores.

adicionaZona: Adiciona uma zona à matriz de zonas.
removeZona: Remove uma zona da matriz com base no ID.
listarZonas: Retorna uma string com informações sobre as zonas.

modificaPropriedade: Modifica o valor de uma propriedade de ambiente numa zona específica.
listarPropriedades: Retorna uma string com as propriedades de uma zona.

adicionaComponente: Adiciona um componente (sensor ou aparelho ou processador) a uma zona.
listarComponentes: Retorna informações sobre os componentes de uma zona.
removeComponente: Remove um componente de uma zona.

novaRegra: Adiciona uma nova regra a um processador de regras numa.
listaRegras: Retorna informações sobre as regras associadas a um processador numa zona.
removeRegra: Remove uma regra de um processador numa zona.
mudaComandoProcessador: Modifica o comando associado a um processador numa zona (liga para desliga ou desliga para liga).

avancaInstante: Avança para o próximo instante de tempo.

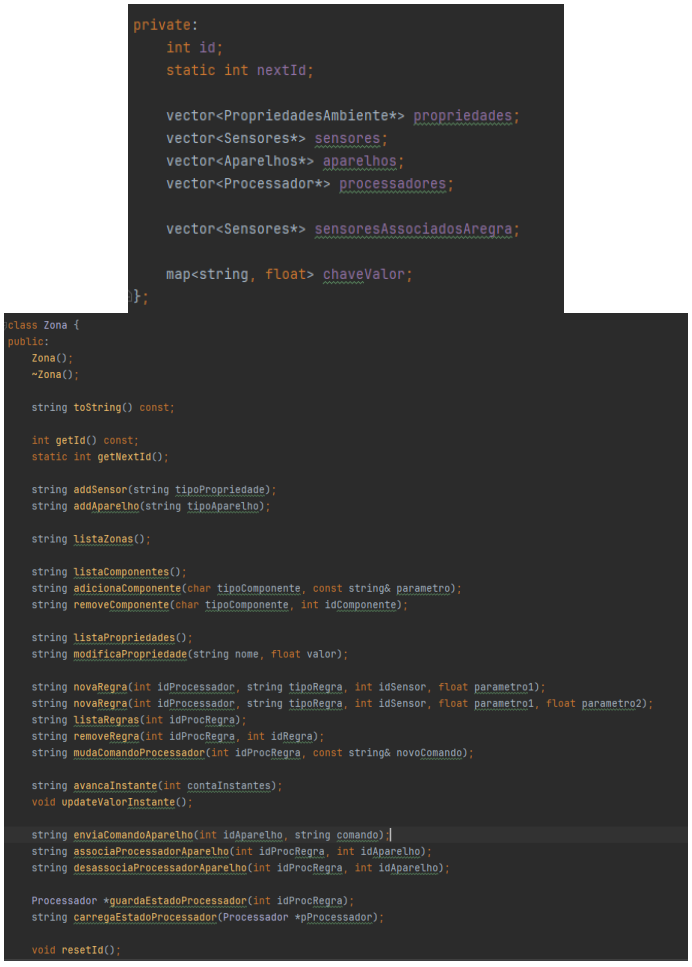
enviaComandoAparelho: Envia um comando para um aparelho numa zona.
associaProcessadorAparelho: Associa um processador a um aparelho numa zona.
desassociaProcessadorAparelho: Desassocia um processador de um aparelho numa zona.

guardaEstadoProcessador: Guarda o estado de um processador em memória.
listaProcessadoresMemoria: Lista os processadores guardados em memória.
carregaEstadoProcessador: Carrega o estado de um processador guardado em memória.
apagaProcessadorMemoria: Apaga um processador da memória.

resetId: Reinicializa os IDs das zonas.

Zona.h

Engloba a representação e a lógica de uma zona dentro do simulador. Fornece métodos para adição e remoção de componentes, manipulação de propriedades, regras e simulação do tempo. Esta classe é essencial para o funcionamento adequado do sistema.



```
private:
    int id;
    static int nextId;

    vector<PropriedadesAmbiente*> propriedades;
    vector<Sensores*> sensores;
    vector<Aparelhos*> aparelhos;
    vector<Processador*> processadores;

    vector<Sensores*> sensoresAssociadosAregra;

    map<string, float> chaveValor;
};
```

```
class Zona {
public:
    Zona();
    ~Zona();

    string toString() const;

    int getId() const;
    static int getNextId();

    string addSensor(string tipoPropriedade);
    string addAparelho(string tipoAparelho);

    string listaZonas();

    string listaComponentes();
    string adicionaComponente(char tipoComponente, const string& parametro);
    string removeComponente(char tipoComponente, int idComponente);

    string listaPropriedades();
    string modificaPropriedade(string nome, float valor);

    string novaRegra(int idProcessador, string tipoRegra, int idSensor, float parametro1);
    string novaRegra(int idProcessador, string tipoRegra, int idSensor, float parametro1, float parametro2);
    string listaRegras(int idProcRegra);
    string removeRegra(int idProcRegra, int idRegra);
    string mudaComandoProcessador(int idProcRegra, const string& novoComando);

    string avancaInstante(int contaInstantes);
    void updateValorInstante();

    string enviaComandoAparelho(int idAparelho, string comando);
    string associaProcessadorAparelho(int idProcRegra, int idAparelho);
    string desassociaProcessadorAparelho(int idProcRegra, int idAparelho);

    Processador *guardaEstadoProcessador(int idProcRegra);
    string carregaEstadoProcessador(Processador *pProcessador);

    void resetId();
```

Map<string,float> chaveValor: Map que armazena pares chave-valor, representando as propriedades específicas da zona.

toString:
Retorna uma representação string da zona.

getId: Retorna o ID da zona.

getNextId: Retorna o próximo ID da zona.
listaZonas: Retorna uma string com informações sobre as zonas.

addSensor: Adiciona um novo sensor à zona.
addAparelho: Adiciona um novo aparelho à zona.

listaComponentes: Retorna informações sobre os componentes da zona.
adicionaComponente: Adiciona um componente (sensor ou aparelho ou processador) à zona.
removeComponente: Remove um componente da zona.

listaPropriedades: Retorna informações sobre as propriedades da zona.
modificaPropriedade: Modifica o valor de uma propriedade específica.

novaRegra: Adiciona uma nova regra a um processador na zona.
listaRegras: Retorna informações sobre as regras associadas a um processador na zona.
removeRegra: Remove uma regra de um processador na zona.
mudaComandoProcessador: Modifica o comando associado a um processador na zona.

avancaInstante: Avança para o próximo instante de tempo.
updateValorInstante: Atualiza os valores relativos ao instante de tempo na zona.

enviaComandoAparelho: Envia um comando para um aparelho na zona.

associaProcessadorAparelho: Associa um processador a um aparelho na zona.
desassociaProcessadorAparelho: Desassocia um processador de um aparelho na zona.

guardaEstadoProcessador: Guarda o estado de um processador da zona.
carregaEstadoProcessador: Carrega o estado de um processador na zona.

resetId: Reinicializa os IDs das zonas.

Aparelhos.h

A classe aparelhos e as suas classes derivadas representam os diferentes tipos de aparelhos que podem existir numa determinada zona.

```
class Aparelhos {
private:
    string nome;
    char letra;
    string estado;
    int id;
    static int nextId;

    bool lastCommand;

public:
    Aparelhos(string tipoAparelho, char letra, string estado);
    ~Aparelhos();

    static int getNextId();

    string getNome() const;
    char getLetra() const;

    string getEstado() const;
    void setEstado(string estado);

    int getId() const;

    string getLastCommand() const;

    virtual void avancaInstante(int contaInstantes);

    void resetId();
};
```

getNome: Retorna o nome do aparelho.
getLetra: Retorna a letra identificadora do aparelho.
getEstado: Retorna o estado atual do aparelho.
setEstado: Define o estado atual do aparelho.
getId: Retorna o ID do aparelho.
getLastCommand: Retorna o último comando recebido.

virtual avancaInstante:
Atualiza o estado do aparelho conforme a passagem do tempo.
resetId:
Reinicializa os IDs dos aparelhos.

```
class Aquecedor : public Aparelhos {
public:
    Aquecedor(map <string,float> *valorPropriedade);
    ~Aquecedor();

    void avancaInstante(int contaInstantes) override;
private:
    bool fezRuido;
    map <string,float> *valorPropriedade;
};
```

bool fezRuido: Indica se o aquecedor fez ruído.
map<string,float>*valorProriedade: Ponteiro para um map representando as propriedades do aquecedor.

avancaInstante (Override):
Atualiza o estado do aquecedor conforme a passagem do tempo.

```
class Aspersor : public Aparelhos {
public:
    Aspersor(map <string,float> *valorPropriedade);
    ~Aspersor();

    void avancaInstante(int contaInstantes) override;

private:
    map <string,float> *valorPropriedade;
    int guardaInstante;
    int contaDesligado;
};
```

map<string,float>*valorProriedade: Ponteiro para um map representando as propriedades do aspersor.

int guardaInstante: Armazena o instante em que o aspersor foi ativado pela última vez.
int contaDesligado: Contador para saber quanto tempo o aspersor permaneceu desligado.

avancaInstante (Override):
Atualiza o estado do aspersor conforme a passagem do tempo.

```
class Refrigerador : public Aparelhos {
public:
    Refrigerador(map <string,float> *valorPropriedade);
    ~Refrigerador();

    void avancaInstante(int contaInstantes) override;

private:
    map <string,float> *valorPropriedade;
    bool addRuido;
};
```

map<string,float>*valorProriedade: Ponteiro para um map representando as propriedades do refrigerador.
bool addRuido: Indica se o refrigerador adicionou ruído.

avancaInstante (Override):
Atualiza o estado do refrigerador conforme a passagem do tempo.

```
class Lampada : public Aparelhos {
public:
    Lampada(map <string,float> *valorPropriedade);
    ~Lampada();

    void avancaInstante(int contaInstantes) override;

private:
    map <string,float> *valorPropriedade;

    bool ligado;
};
```

map<string,float>*valorProriedade: Ponteiro para um map representando as propriedades da lâmpada.
bool ligado: Indica se a lâmpada está ligada.

avancaInstante (Override):
Atualiza o estado da lâmpada conforme a passagem do tempo.

Sensores.h

Classe que representa os sensores presentes na zona. Cada sensor pode ler uma propriedade específica e fornecer o seu valor atual.

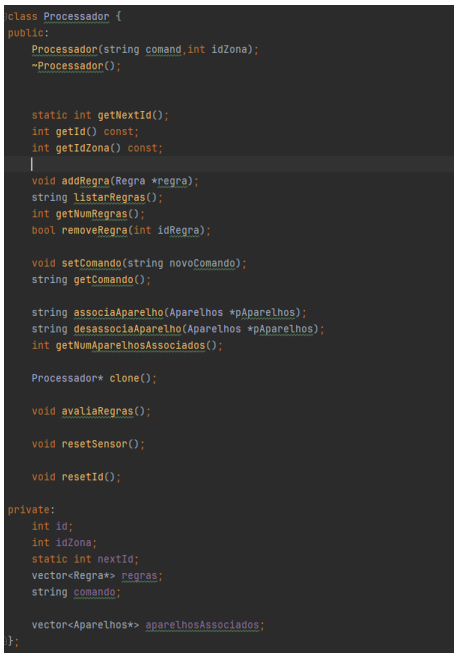
```
class Sensores {
private:
    char letra;
    string nome;
    string propriedade;
    int id;
    static int nextId;
    map<string,float> *chaveValor;

public:
    Sensores(char letra,string propriedade, map<string,float> *chaveValor);
    ~Sensores();

    static int getNextId();
    int getId() const;

    string getNome() const;
    char getLetra() const;

    float getValorAtual();

    void resetId();
};
```

map<string,float> *chaveValor: Ponteiro para um map que armazena as propriedades associadas ao sensor e seus valores.

getNextId: Retorna o próximo ID disponível para atribuição.
getId: Retorna o ID do sensor.
getNome: Retorna o nome do sensor.
getLetra: Retorna a letra identificadora do sensor.
getValorAtual: Retorna o valor atual associado ao sensor.
resetId: Reinicializa os IDs dos sensores.

Processadores.h

Classe que gere as regras e que permite que as ações sejam tomadas com base nas condições definidas pelas regras.



```
class Processador {
public:
    Processador(string comand,int idZona);
    ~Processador();


    static int getNextId();
    int getId() const;
    int getIdZona() const;

    void addRegra(Regra *regra);
    string listarRegras();
    int getNumRegras();
    bool removeRegra(int idRegra);

    void setComando(string novoComando);
    string getComando();

    string associaAparelho(Aparelhos *pAparelhos);
    string desassociaAparelho(Aparelhos *pAparelhos);
    int getNumAparelhosAssociados();

    Processador* clone();

    void avaliaRegras();

    void resetSensor();

    void resetId();

private:
    int id;
    int idZona;
    static int nextId;
    vector<Regra*> regras;
    string comando;

    vector<Aparelhos*> aparelhosAssociados;
};
```

static int getNextId(): Retorna o próximo ID disponível para atribuição a novos processadores.
int getId() const: Retorna o ID do processador.
int getIdZona() const: Retorna o ID da zona à qual o processador está associado.
void addRegra(Regra *regra): Adiciona uma nova regra ao processador.
string listarRegras(): Retorna uma string com informações sobre as regras associadas ao processador.
int getNumRegras(): Retorna o número de regras associadas ao processador.
bool removeRegra(int idRegra): Remove uma regra do processador com base no ID da regra.
void setComando(string novoComando): Define um novo comando para o processador.
string getComando(): Retorna o comando associado ao processador.
string associaAparelho(Aparelhos *pAparelhos): Associa um aparelho ao processador.
string desassociaAparelho(Aparelhos *pAparelhos): Desassocia um aparelho do processador.

int getNumAparelhosAssociados(): Retorna o número de aparelhos associados ao processador.
Processador* clone(): Cria uma cópia do processador.
void avaliaRegras(): Avalia as regras associadas ao processador.
void resetSensor(): Reinicializa os sensores associados ao processador.
void resetId(): Reinicializa os IDs dos processadores.

Regra.h

Permite definir as condições para os sensores.

```
class Regra {
public:
    Regra(Sensores &sensor, float x,string nome);
    Regra(Sensores &sensor, string nome);
    int getId() const;
    string nome;

    bool verificar();

    float x;

    Sensores * getSensor();
    void resetSensor();

    string getTipo();


    virtual float getX();
    virtual float getY();

    void resetId();

private:
    int id;
    static int nextId;

protected:
    Sensores *sensor;
};
```

getId() const: Retorna o ID da regra.
verificar(): Verifica se a condição da regra é atendida.
getSensor(): Retorna o ponteiro para o sensor associado à regra.
resetSensor(): Reinicializa o sensor associado à regra.
getTipo(): Retorna o tipo da regra.
resetId(): Reinicializa os IDs das regras.

```
class RegraEntre : public Regra {
public:
    RegraEntre(Sensores &sensor, float x, float y);
    bool verificar();
    float getX() override;
    float getY() override;
private:
    float x;
    float y;
};

class RegraFora : public Regra {
public:
    RegraFora(Sensores &sensor, float x, float y);
    bool verificar();
    float getX() override;
    float getY() override;

private:
    float x;
    float y;
};
```

verificar(): Verifica se a condição da regra é atendida.
getX(): Retorna o primeiro valor associado à regra.
getY(): Retorna o segundo valor associado à regra.

PropriedadesAmbiente.h

Representa as propriedades de ambiente que são controladas pelos componentes.

```
class PropriedadesAmbiente {
private:
    string nome;
    string unidade;
protected:
    float min;
    float max;
    float atual;
public:
    PropriedadesAmbiente(string nome, string unidade, float min, float max, float atual);
    ~PropriedadesAmbiente();

    string getNome() const;
    string getUnidade() const;
    float getMin() const;
    float getMax() const;
    float getAtual() const;

    virtual void alteraValor(float valor);

    string toString();
};
```

getNome() const: Retorna o nome da propriedade.
getUnidade() const: Retorna a unidade de medida da propriedade.
getMin() const: Retorna o valor mínimo da propriedade.

getMax() const: Retorna o valor máximo da propriedade.
getAtual() const: Retorna o valor atual da propriedade.
Virtual alteraValor(float valor): Altera o valor atual da propriedade.
toString(): Retorna uma string com informações sobre a propriedade.

```
class Humidade : public PropriedadesAmbiente {
public:
    Humidade(float min, float max, float atual);
    void alteraValor(float valor);
    ~Humidade();

};


class Fumo : public PropriedadesAmbiente {
public:
    Fumo(float min, float max, float atual);
    void alteraValor(float valor);
    ~Fumo();

};
```

alteraValor(float valor): Altera o valor atual da propriedade.

Comandos.h

Interface principal para a interação com o simulador. Engloba os comandos disponíveis e fornece métodos para interagir com a habitação.

```
public:
    Comandos();
    void startSimulator();
private:
    string VerificaComandos(const string& comando);
    bool existeHabitacao = false;
    void setExisteHabitacao(bool valor);
    string avancaInstante(const string& input);
    string criaHabitacao(const string& input);
    string removeHabitacao(const string& input);
    string criaNovaZona(const string& input);
    string removeZona(const string& input);
    string listaZonas(const string& input);
    string listaComponentes(const string& input);
    string listaPropriedades(const string& input);
    string modificaPropriedade(const string& input);
    string adicionaComponente(const string& input);
    string removeComponente(const string& input);
    string novaRegra(const string& input);
    string mudaComandoProcessador(const string& input);
    string listaRegras(const string& input);
    string removeRegra(const string& input);
    string associaProcessadorAparelho(const string& input);
    string desassociaProcessadorAparelho(const string& input);
    string enviaComandoAparelho(const string& input);
    string guardaEstadoProcessador(const string& input);
    string carregaEstadoProcessador(const string& input);
    string apagaProcessadorMemoria(const string& input);
    string listaProcessadoresMemoria(const string& input);
    string carregaFicheiroComandos(const string& input);
    Habitacao *habitacao = nullptr;

    Window *janelaOutput = nullptr;

    vector<Window> windows;
    void criarWindows();
    void refreshHabitacao();
    void clear();
};
```

Opções tomadas

Inicializar a +9999 as propriedades de ambiente que não tenham valor máximo. Inicializar as propriedades com o valor default para o mínimo, caso contrário estavam a ser inicializadas com lixo.

Função updateValorInstante():
Os aparelhos vão modificar as propriedades através do map chaveValor, o problema é que o map estava a ficar atualizado, mas o vetor de propriedades da zona estava desatualizado e tem de ser atualizado o valor de cada objeto Propriedade com o valor que está no map.

Para o simulador reagir à passem do tempo, é sempre preciso +1 instante.

Comando clear para limpar a janela de output a qualquer momento.

Função resetSensor():
Para não guardar os sensores quando guardo o processador em memória.

Funcionalidades

Implementado parcialmente:

O sensor pode ser removido caso tenha regras associadas, assim como o aparelho se tiver processadores. O processador não pode ser removido caso tenha regras.

Repor o processador em memória caso já exista um na zona. Apenas é possível repor caso não exista na zona.

Implementado:

Todos os comandos, exceto os comandos que estão incompletos nas funcionalidades parcialmente implementadas.

Aparelhos, Sensores e Processadores, regras e propriedades e toda a ligação entre as classes implementada.

Conclusão

Acreditamos que que o projeto abrange diversos aspetos da programação orientada a objetos crucial para a aprendizagem e para o aprofundar dos conhecimentos. Ao longo do projeto foram abordados vários conceitos fundamentais como agregação, composição, polimorfismo, heranças que refletem uma sólida compreensão dos princípios de C++.

Por último, acreditamos que realizámos um bom trabalho e que atendemos ao que era pedido no enunciado.