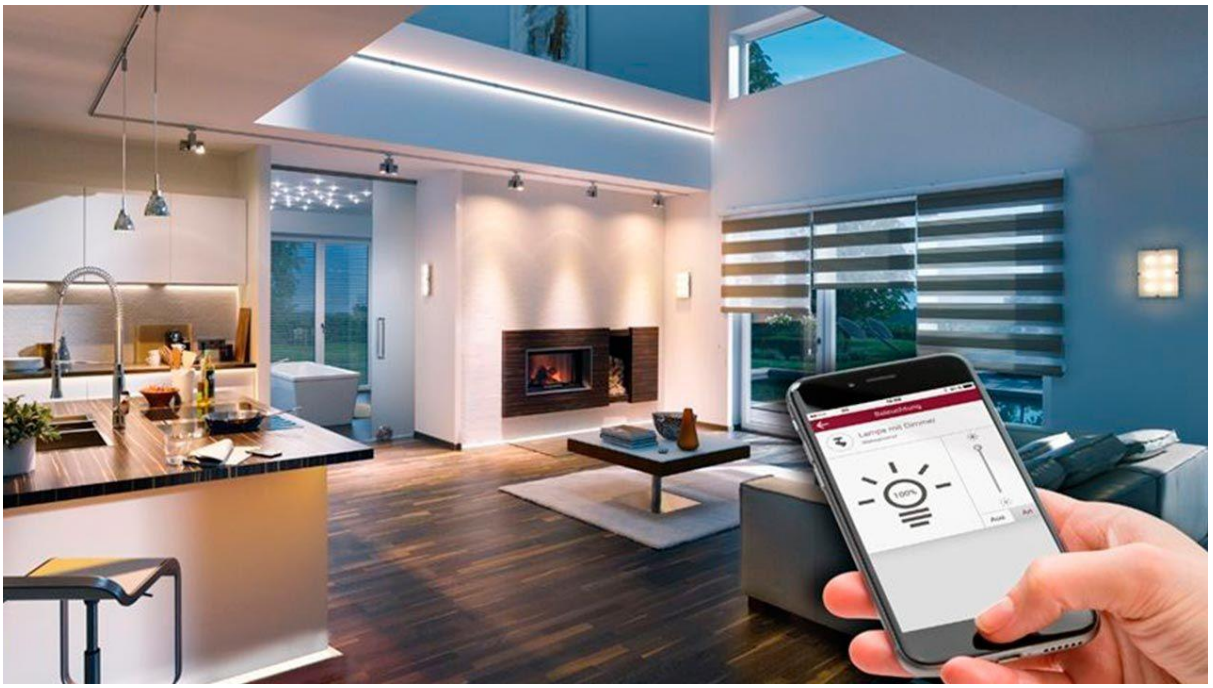
isec
Politécnico de Coimbra

Trabalho Prático

Criação de uma casa automática

P.O.O.



João Afonso Fonseca Costa Almas – 2021138417- a2021138417@isec.pt
Paulo Guilherme Lopes Sá - 2021142819 - a2021142819@isec.pt

2023/2024

Índice

1.Introdução:

O objetivo deste trabalho prático em Programação Orientada a Objetos é desenvolver um simulador residencial interativo que permita aos utilizadores controlar e monitorar diferentes dispositivos numa habitação virtual. O simulador foi concebido para refletir cenários do mundo real, proporcionando uma experiência de utilização prática.

O problema abordado neste projeto envolve a criação de um ambiente simulado onde os habitantes da casa podem interagir com aparelhos e sensores diversos. O simulador busca simular a dinâmica de uma casa inteligente, onde os utilizadores podem ligar e desligar dispositivos, ajustar configurações e observar como as mudanças afetam o ambiente virtual.

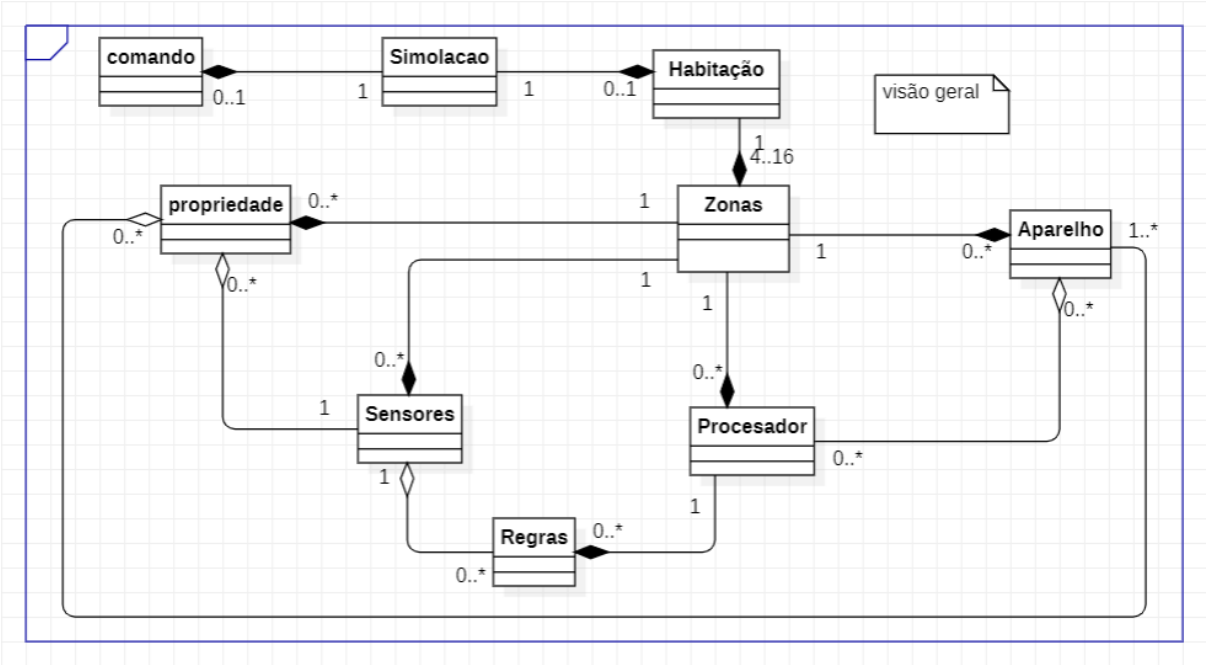
Objetivos do Trabalho:

- Desenvolver um sistema robusto e flexível utilizando os princípios da Programação Orientada a Objetos.
- Implementar sensores, processadores de regras e aparelhos, explorando os conceitos de encapsulamento, herança e polimorfismo.
- Criar uma interface de utilizador intuitiva.
- Demonstrar a aplicação prática de conceitos aprendidos em Programação Orientada a Objetos no contexto do desenvolvimento de software.

Este relatório fornecerá uma visão abrangente da arquitetura do simulador, detalhes de implementação, justificativas de design e exemplos de casos de uso. No final, espera-se que o leitor tenha uma compreensão sólida do funcionamento do simulador e das decisões tomadas ao longo do processo de desenvolvimento.

2. Arquitetura do Simulador:

Neste ponto, abordaremos a estrutura geral do simulador, destacando as principais classes e sua interconexão. A arquitetura do software é crucial para garantir a eficiência, modularidade e escalabilidade do sistema. A seguir, serão discutidos os principais componentes do simulador:



Class UI(*User Interface*):

Composição da Class **UI**: Tem uma *class* Comando e uma *class* Habitação. Os atributos estão todos encapsulados com private, e estão presentes os métodos get e set necessários. A função desta class é coordenar o uso da *class* comando para determinar qual comando que o utilizador está a utilizar, realizar alterações na interface gráfica do programa e comunicar à class habitação essas mesmas alterações.

Como possui alocação de memória dinâmica, a responsabilidade de destruição do objeto é da *class* **UI**.

Class Comando:

Sem composição ou agressão: A class **Comando** tem a função de analisar e identificar o comando que foi inserido, evitando erros de sintaxe e a falta de argumentos.

Class Habitação:

A class **Habitação** tem composição sobre os objetos **Zona**, que são no mínimo 4 e no máximo 16, e objetos **Processador** (processadores de regras), que estão armazenados na "memoria" para que o utilizador possa repor quando quiser.

Como a class Habitação está a utilizar smart-pointers para os objetos Processadores, só é necessário destruir os objetos **Zona**, pois os smart pointers cuidarão automaticamente da gestão de memória dos objetos **Processador**.

Class Zona:

A class **Zona** tem composição sobre objetos **Propriedade**, **Aparelho**, **Sensor** e **Processador**. As **Propriedades** estão armazenadas num template map, com a chave sendo o nome da propriedade e o valor associado. Os Aparelhos, Sensores e Processadores estão armazenados num *tamplate* vector e é utilizado polimorfismo para permitir vários tipos de aparelhos num único vector de aparelhos. Foi escolhido o uso de smart pointers para todos os objetos que a class Zona possui.

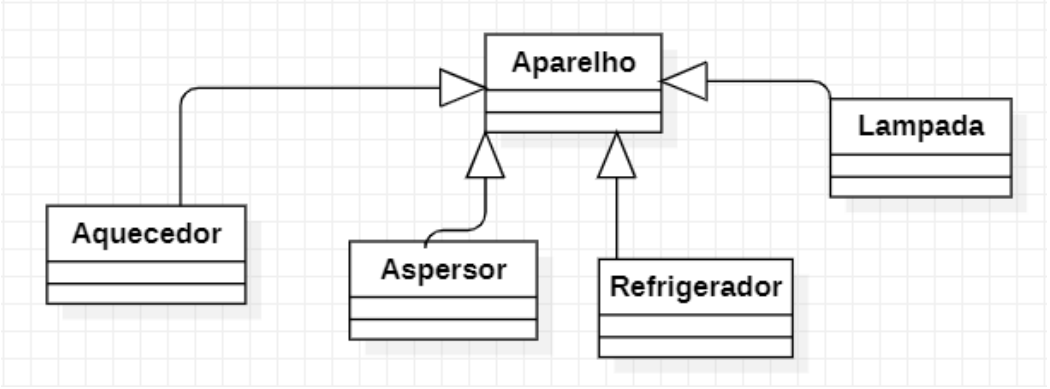
Class Propriedade:

A class **Propriedade** não possui nem agregação nem composição. Como esta class foi feita na primeira meta, quando ainda não tinha sido lecionada matéria sobre herança, decidimos não implementar esse conceito uma vez que já se encontrava funcional sem o mesmo.. No entanto, agora temos a visão de que poderíamos melhorar a class com herança (class PropriedadeBase, PropriedadeComMinimo, PropriedadeComMax e PropriedadeMaxMinimo) e polimorfismo. Esta class desempenha a função de uma propriedade como, por exemplo, temperatura.

Class Sensor:

Os **Sensor** têm apenas agregação, o que proporciona a capacidade de visualizar o valor de um objeto da class Propriedade através dos métodos get.

Class Aparelho:

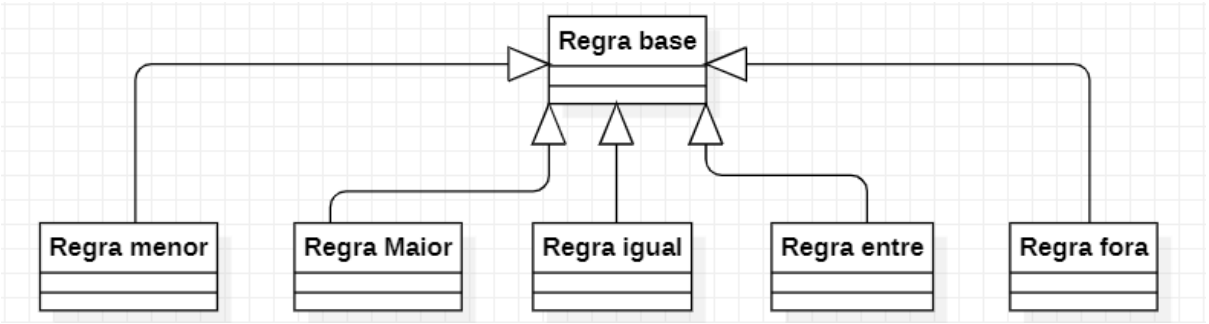


O **Aparelho** contém herança e agregação com as propriedades, utilizando a classe base Aparelho com os métodos necessários, como virtuais, para proporcionar a capacidade de alterar o método das derivadas. As classes derivadas, que são Aquecedor, Aspersor, Lâmpada e Refrigerador, recebem um comando e, se tiverem maneira de lidar com esse comando, como por exemplo, ligar, alteram algumas propriedades cada uma à sua maneira distinta.

Class Processador:

A *class* **Processador** possui composição em relação aos objetos Regra e agregação em relação aos objetos Aparelhox. A *class* verifica as regras, e se todas forem verdadeiras, envia o comando para os aparelhos associados a esse processador. Ele avalia apenas as regras que têm um sensor associado, portanto, se uma regra for criada e depois o sensor for eliminado, essa regra será ignorada na avaliação no estado do processador.

Class Regra:



A *class* **Regra** desfruta da herança, tendo 5 derivadas onde cada uma possui algumas funcionalidades diferentes. Existe agregação com o Sensor, onde pode visualizar o valor que o sensor fornece, proporcionando basicamente uma janela para uma propriedade.

Todas as classes, a partir da Zona, utilizam apenas smart pointers para composição, sendo unique_ptr e shared_ptr. Para agregação, são utilizados apenas weak_ptr. este programa é facilmente escalável devido à aplicação das práticas de Programação Orientada a Objetos (POO).

3. Conceitos de Programação Orientada a Objetos Aplicados

Todas as classes têm implementado o conceito de **encapsulamento**, os seus atributos são privados. O conceito de **herança** está presente nas classes aparelho e regra. Usámos polimorfismo nas *class* zona para os aparelhos e também na *class* processador com as regras para facilitar libertação de memória dinamicamente alocada no final do programa no caso de este ser terminado de maneira regular. Os pilares da Programação Orientada a Objetos foram postos em prática como **Encapsulamento**, **Abstração**, **Herança** e **Polimorfismo**.

4. Manual do Utilizador

Comandos para o tempo simulado

1. prox : avança um instante.
2. avanc <n> : avança “n” instante.

Comandos para gerir habitação e zonas

1. hnova <num linhas> <num colunas> :
   número de linhas são 2 - 4, número de camada são 2 - 4.
2. hrem : apaga a habitação.
3. znova <linha> <coluna> : começa no znova 0 0.
4. zrem <ID zona> : apagar a zona como o id.
5. zlista : listar zonas.

Comandos para gerir zonas e seu conteúdo

1. zcomp <ID zona> : mostrar info de uma zona.
2. zprops <ID zona> : mostrar info das propriedades.
3. pmod <ID zona> <nome> <valor> : mudar um valor da propriedades.
4. cnovo <ID zona> <s | p | a> <tipo | comando> : criar um aparelho ou sensor ou processador. Aparelho que pode criar sao (Aquecedor, Aspersor, Refrigerador, Lampada);
5. crem <ID zona> <s | p | a> <ID> : remover um aparelho ou sensor ou processador.

comandos para processadores de regras

1. rnova <ID zona> <ID proc. regras> <regra> <ID sensor> [param1] [param2] [...] : criar regras. regras que podes criar (igual, maior, menor, entre, fora).
2. pmuda <ID zona> <ID proc. regras> <novo comando>: mudar comando do processador, As Propriedade que está inicializada podem ser vistas no comando zprops.
3. rlista <ID zona> <ID proc. regras> : listar regras de um processador.
4. rrem <ID zona> <ID proc. regras> <ID regra> : eliminar uma regra.
5. asoc <ID zona> <ID proc. regras> <ID aparelho> associar um aparelho a um procesador.
6. ades <ID zona> <ID proc. regras> <ID aparelho> : remover a associação um aparelho a um processador.

Comandos para interagir com aparelhos

1. acom <ID zona> <ID aparelho> <comando> : mandar um comando manual para um aparelho.

Comandos para cópia/recuperação de processadores de regras

1. psalva <ID zona> <ID proc. regras> <nome> : salvar um processador.
2. prepoe <nome> : repor um processador salvo na zona que ele foi retirado.
3. prem <nome> : remover um processador salvo .
4. plista : listar processador salvo.

Comandos adicionais de carácter geral

1. exec <nome de ficheiro> : executar um txt com comandos.
2. sair : sair com programa.

Estes são os nomes dos comandos que você pode utilizar no seu programa. Certifique-se de seguir a sintaxe correta para cada comando.

Conclusão:

O trabalho prático proporcionou uma experiência prática na aplicação dos princípios da Programação Orientada a Objetos. A criação do simulador, com seus diversos componentes e interações entre eles, destacou a importância dos pilares deste paradigma de programação, como encapsulamento, herança e polimorfismo.

A implementação das classes e a manipulação de objetos demonstraram a eficácia do encapsulamento na gestão de componentes. A herança facilitou a criação de diversos

tipos de componentes, promovendo a reutilização de código. O polimorfismo contribuiu para tratar objetos de maneira uniforme, aumentando a flexibilidade do simulador.

Enfrentamos desafios que exigiram análise crítica e soluções criativas, especialmente na interação entre componentes e na aplicação de regras. O projeto serviu como uma base sólida para futuros desenvolvimentos em programação orientada a objetos e simulações de sistemas complexos.