UNIVERSIDADE FEDERAL DA PARAÍBA – CENTRO DE INFORMÁTICA

LPII - Programação Orientada a Objetos - 2024.2

Prof. Carlos Eduardo Batista

Discente: Rodolfo França de Souza

Relatório

São três classes principais que envolvem funcionalidade: a classe base abstrata dispositivo\_industrial, que contém os métodos virtuais puros que serão sobrescritos pelas classes descendentes controlador\_motor, robo\_manipulador e sensor\_temperatura. Na implementação, cada uma tem seu arquivo com cabeçalho, e também implementei outro arquivo com funções de utilidade, chamado utils.cpp, que contém uma função para escrever o log em um arquivo de texto utilizando ofstream, chamada escreverLog, e também a função lerLog, que acabei não utilizando. Em utils.cpp, também tem uma função para obter a hora, que implementei com as bibliotecas padrões chrono, ctime e sstream, a qual retorna a hora, minutos e segundos em uma string.

Na classe base dispositivo\_industrial, utiliza-se o atributo string id para armazenar a string, status, que vai indicar se o dispositivo está para ser executado ou desligado (isso irá trazer problemas que comentarei depois), e temperatura. Com um construtor básico para id e temperatura, e um destruidor virtual. Getters e setters para cada atributo e os métodos virtuais requeridos no documento.

A primeira classe derivada, controlador\_motor, tem os atributos int potencia, int rpm, double horas\_de\_uso, string relatorio, string warning. Os dois últimos atributos foram adicionados para complementar a função gerar\_relatorio(). Possui um construtor que pede como parâmetros o id, a temperatura, a potência e as rotações por minuto. Em ajustar\_velocidade, a máquina verifica se o rpm passado como parâmetro é maior que 0; se sim, salva para o relatório e atribui à variável rpm o valor de nova\_rpm. Caso contrário, não atribui e faz um log no relatório. A função calcular\_eficiencia() faz uma inferência para uma percepção fictícia da qualidade da eficiência do gasto energético (potência) em relação ao rpm. O método verificar\_seguranca() é um método booleano override que verifica se a potência ou o rpm estão maiores que o número estabelecido. Se estiverem extrapolando os limites, ele retorna true; caso contrário, retorna false. Isso é utilizado na função iniciar(), caso os limites sejam extrapolados, a máquina será parada através da função parar(). A função gerar\_relatorio() informa o nome do id e o tempo de utilização da máquina em segundos, contados através do loop while de iniciar().

A classe sensor\_temperatura tem atributos derivados e alguns adicionais, como double temperatura\_maxima, double temperatura\_minima, double temp\_atual e string relatorio. O construtor é simples e atende aos requisitos da classe base e da classe sensor\_temperatura, como as temperaturas mínima e máxima, e utiliza a temperatura da classe dispositivo\_industrial. Ela também verifica se a temperatura máxima é maior ou igual à mínima. O método específico alerta\_temperatura() pega a temperatura atual e verifica se ela é maior que a temperatura mínima e, ao mesmo tempo, menor que a temperatura máxima. Caso sim, registra isso no relatório e no console e retorna false. Caso contrário, se a temperatura atual for maior que a temperatura máxima ou mais baixa que a temperatura normal de funcionamento, ele retorna true. No método iniciar(), ele indica o id, coloca no relatório e informa a temperatura, além de rodar a função específica da classe alerta\_temperatura() em loops de 1 segundo, que são validados através do getStatus. O método parar() coloca o setStatus(false) e gera o relatório.

A classe derivada robo\_manipulador herda de dispositivo\_industrial e indica, em seus atributos, a posição do manipulador, que se inicia com 0 em cada um dos três eixos, sua carga máxima e os atributos auxiliares de relatorio, string warning e string relatorio. Seu construtor pede como parâmetros a referência string id, a temperatura, a carga máxima e mínima. Tem como métodos específicos mover\_para, que pede como parâmetros as coordenadas, e agarrar\_objeto, que verifica se o peso é maior que a carga máxima. O método iniciar() segue o mesmo padrão das outras funções de iniciar, mas com a verificação de segurança (verificar\_seguranca()) para checar se a carga atual é maior que a máxima. Também há a função soltar\_objeto, que solta a carga que está sendo carregada.

A classe sistema\_de\_controle não herda de ninguém e tem vetores de ponteiros de dispositivos\_industriais e de vetores de threads. Ela possui a função adicionar\_dispositivo, que adiciona um novo dispositivo como ponteiro ao vetor dispositivos. A função remover\_dispositivo busca e remove um dispositivo do vetor com base no seu ID. Caso encontrado, ele é deletado; caso contrário, exibe uma mensagem de erro. O método iniciar() tenta iniciar cada dispositivo em uma thread, mas isso está dando problema porque falta experiência para desenvolver um sistema concorrente. O método verificar\_seguranca() pede a cada dispositivo para rodar a função verificar\_seguranca, assim como o método gerar\_relatorio\_completo, que faz a mesma coisa. O destruidor tenta encerrar as threads e dar delete em cada dispositivo.

Tive dificuldade na hora de rodar o loop while, que ficasse ligado somente enquanto o status fosse positivo. Encontrei uma forma de resolver isso com threads, mas a implementação está visivelmente precária. Tentei ainda usar outras soluções, como um botão para parar de rodar o loop com ncurses, mas estava cada vez pior, então fiquei apenas com essa implementação básica de thread mesmo. Na main, tentei criar, dessa forma, os testes de cada um dos dispositivos, testando suas temperaturas, movimentos e cargas, e também funciona parcialmente adicionar os dispositivos através do sistema iniciar\_todos, só que o output do console fica completamente quebrado. A parte de herança, métodos virtuais puros e polimorfismo deu para ser bem elucidada.

Para compilar, é só rodar make na raiz da pasta, já existe um makefile com o comando de compilação.