CAPÍTULO 2

Introdução

"As três coisas mais difíceis no mundo: guardar segredo, perdoar uma injúria e aproveitar o tempo"
— Benjamin Franklin

2.1 - INTRODUÇÃO

Considere o problema de descobrir a altura da pessoa mais alta de um grupo de pessoas. Suponha que estas pessoas estão em seqüência, como em uma fila de banco, e que esta fila não está vazia.

Vamos elaborar uma estratégia para resolver este problema. Uma solução bem simples seria fazer o seguinte:

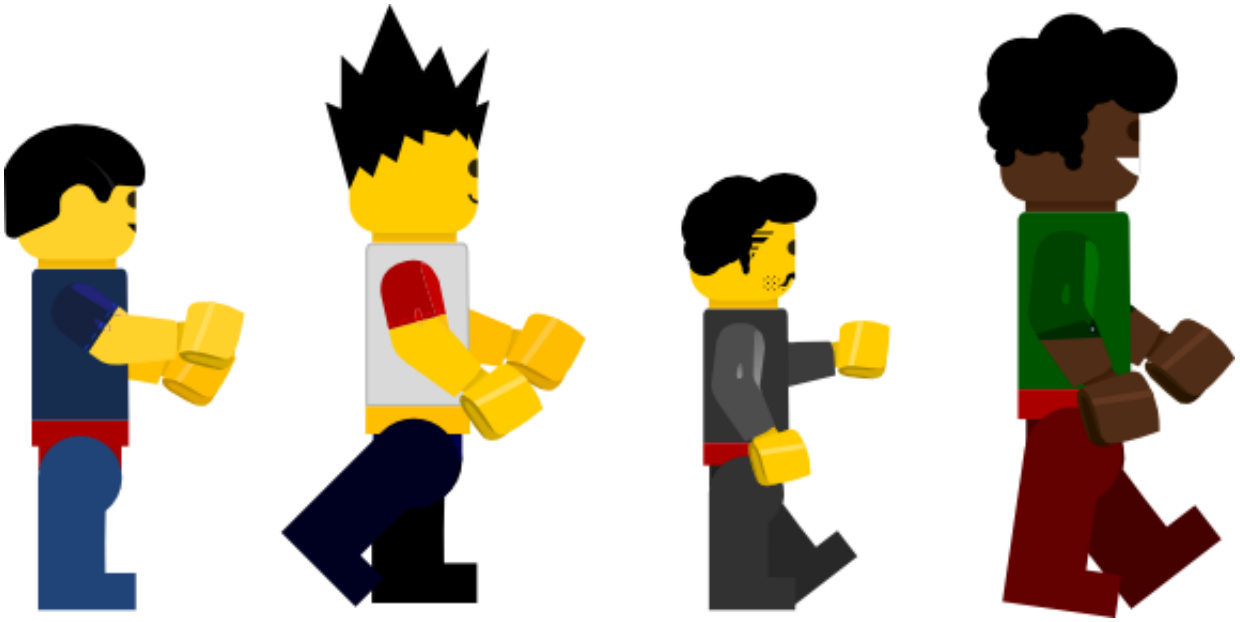

Figura 2.1: Fila de pessoas

1. Pegue a altura da primeira pessoa. A única **informação** que você tem é que esta altura é a máxima até o momento. Então, "guarde" em algum lugar esta informação.

2. Percorra cada uma das próximas pessoas e faça o seguinte:

    1. Pegue a altura da pessoa, esta é a altura atual.

    2. Compare a altura atual com a máxima até o momento. Esta comparação pode resultar em três possibilidades: a altura atual é menor, é igual ou é maior.

    3. Se a altura atual for maior, então faça o valor da altura máxima ser igual a atual.

Será que este procedimento é bom? Nesse caso temos de percorrer todas as pessoas da fila até ter certeza que encontramos a maior pessoa, pois a única invariante que temos, a cada passo do procedimento, é que até aquele momento todas as pessoas percorridas tem tamanho igual ou menor a um determinado número.

Existe solução melhor? E se tivéssemos as pessoas organizadas de alguma maneira especial, chegaríamos a solução em um menor tempo? E se houveram empates, quais outras pessoas são tão altas quanto essa nossa resposta?

2.2 - Algoritmo e Implementação

Um **algoritmo** é uma seqüência de passos que resolve algum problema ou alcança algum objetivo, como a seqüência de passos para resolver o problema de descobrir a máxima altura. É importante salientar que um algoritmo simplesmente diz o que **deve** ser feito.

Para resolver de fato um problema, devemos definir **como** executar os passos do algoritmo. Por exemplo, para o problema anterior de achar a máxima altura, deveríamos definir como "pegar" as informações sobre as alturas das pessoas (perguntar para a própria pessoa, medir a altura usando uma fita métrica ou obter a altura de algum cadastro que a pessoa tenha feito) e como manter as informações sobre as alturas (anotar em um papel ou guardar em uma variável no computador).

A definição de como os passos de um algoritmo serão executados é uma implementação do algoritmo. Resumindo, algoritmo é o que deve ser feito e implementação é o como deve ser feito.

Estamos interessados em desenvolver algoritmos computacionais. Para isso, utilizaremos um modelo de programação. Um modelo de programação fornece idéias e conceitos para nos ajudar a criar algoritmos. Neste curso, será utilizado o paradigma da programação orientado a objetos (**OO**).

Os nossos algoritmos serão executados por um computador. Então, devemos implementá-lo através de programas de computador. Um programa é a definição de como os passos de um algoritmo serão executados no computador.

Os programas são escritos em alguma linguagem de programação. Uma linguagem de programação é a maneira de "conversarmos" com um computador. A linguagem que utilizaremos aqui é a **Java**. Esta linguagem é voltada para o paradigma de programação orientado a objetos.

2.3 - Estrutura de Dados

Hoje em dia, a grande maioria das pessoas utilizam a agenda do celular para armazenar seus contatos. As tarefas de uma agenda de contatos são basicamente duas:

1. Definir como as informações dos contatos serão armazenadas. Uma informação armazenada em algum lugar (pedaço de papel, livro, computador, etc) é um **dado**.

2. Disponibilizar operações para criar, recuperar, ordenar, atualizar e remover contatos. Além de operações para informar o estado da agenda (a quantidade de contatos existentes na agenda ou a quantidade de espaço disponível para novos contatos).

A primeira tarefa é crucial para o desempenho. Se a agenda armazena as informações de uma forma desorganizada então será muito mais complicado manipular os dados de forma eficiente. A escolha de como guardar as informações

deve levar em consideração as operações que serão disponibilizadas pela agenda. Por exemplo, seria interessante manter os contatos em ordem alfabética para facilitar a busca.

Mas, apesar da importância de como os contatos são armazenados, a organização interna da agenda não precisa e não deve ser exposta ao usuário. Afinal de contas, o que o usuário deseja é usar a agenda através das operações e que tudo seja feito o mais rápido possível.

A única coisa que precisa ser mostrada para o usuário são as operações que ele pode fazer na agenda (inserir, recuperar, atualizar, remover contato, saber quantos contatos estão na agenda, etc). Este conjunto de operações é a **interface** que o usuário tem com a agenda.

Cada celular pode implementar a sua agenda de contatos de uma forma totalmente diferente um do outro, na tentativa de obter mais performance, ser mais confiável ou gastar menos memória. Porém o conjunto básico de operações oferecidas pelas agendas é sempre o mesmo. Isso facilita a vida do usuário pois se ele tiver que trocar de celular não vai ter que aprender novamente como usar uma agenda de contatos.

Essa é a grande vantagem em se pensar em interface. Mantida a interface, podemos trocar uma agenda que não é tão eficiente ou que já não atende mais as nossas necessidades por outra mais eficiente ou adequada, sem problemas em ter de aprender a usar a nova agenda: troca-se a implementação, mas não mudamos a interface.

Uma agenda de celular pode ser vista como uma **estrutura de dados**. Uma estrutura de dados mantém os dados organizados seguindo alguma lógica e disponibiliza operações para o usuário manipular os dados.

É importante, quando programar, não misturar dado e estrutura de dados em uma coisa só. Um dado é uma informação armazenada e estrutura de dados é quem administra os dados. O ideal é que a estrutura de dados seja o mais independente possível dos dados que ela vai armazenar. Desta forma pode-se aproveitar a mesma estrutura de dados para diversos tipos de dados. Por exemplo, é melhor ter uma agenda genérica do que uma agenda de telefones. Uma agenda genérica pode ser utilizada para guardar telefones, ou emails, ou até mesmo guardar uma outra estrutura dentro dela: contatos, que seriam compostos por nome, telefone e email.

Algumas estruturas de dados são apenas agrupamentos de dados sem um

objetivo de aplicar algum algoritmo ou tirar proveito de sua estrutura. Um exemplo seria a estrutura **Contato**. Algumas outras estruturas são mais espertas e trabalhosas, como a **Agenda**, assim como Listas Ligadas, Vetores, Tabelas de Espalhamento e outras que veremos no decorrer do texto. Estas estruturas, por sua característica mais complexa e de poder ser reutilizada em outros contextos, devem ser criadas da forma mais independente possível dos dados que estarão dentro dela. Em outras palavras, não devemos misturar a **Agenda** e o **Contato** de maneira rígida, para que com isso possamos criar outras Agendas, como por exemplo uma Agenda de **Fornecedor**.

2.4 - Sobre este texto

Este texto vai mostrar a você diversas estruturas de dados, suas vantagens e desvantagens, assim como suas implementações básicas e classes já existentes na biblioteca padrão do Java.

Vamos usar recursos do Java como interfaces, generics, exceptions, pacotes e outros. É bom já ter um conhecimento razoável da linguagem e um pouco de orientação a objetos.

CAPÍTULO ANTERIOR:

Prefácio

PRÓXIMO CAPÍTULO:

Armazenamento Sequencial

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter