Herança

A Programação Orientada a Objetos foi criada por Alan Kay e une seus dois principais campos de estudos, a matemática e a biologia. Segundo Kay, o computador ideal deveria se comportar como um organismo vivo, mais próximo do mundo real. Isso quer dizer que, o raciocínio lógico se uniria ao raciocínio biológico, fazendo com que cada coisa fique em seu lugar e se comporte como previamente definido. Ou seja, classes podem funcionar de forma autônoma, mas também se agrupar com outras classes para atingir um objetivo em comum. Além disso, as classes podem se reagrupar para desempenharem outras funções.

Dito isso, a herança desempenha um papel fundamental neste paradigma e por sua vez, é um de seus pilares. Nela é possível que uma classe tenha uma definição base e conceitual de atributos e métodos que podem ser herdados por classes filhas e possuir outras funcionalidades, sem deixar de possuir os traços que a definem como filha da classe mãe. Ou seja, se o sistema possui uma classe que tem como função principal enviar e-mails, é possível criar uma classe que herde seus atributos e funções e a partir daí agregar outras funcionalidades, por exemplo, uma função que envie um e-mail de boas-vindas ou felicitações de aniversário.

O conceito de herança traz a possibilidade de basear a idealização de um objeto estendido a partir de outro existente. Desse modo, classes que possuem atributos e métodos idênticos, podem ser aplicados em uma classe Pai para que apenas suas classes filhas tenham variações.
Utilizaremos como exemplo de declaração a classe Pessoa, onde ela possui atributos como ‘nome’, ‘email’, ‘telefone’, métodos básicos de cadastrar e deletar. Suas classes filhas, ‘Cliente’, ‘Funcionário’ e ‘Fornecedor’ herdam seus atributos e comportamentos, além de terem suas particularidades declaradas posteriormente. Podemos ter uma forma mais visual de como a estrutura se comportaria no sistema, na imagem a seguir:

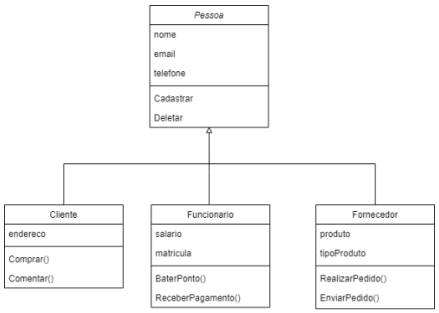


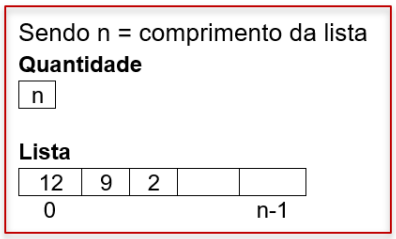
Figura 1.Diagrama de Classe

Herança no contexto de implementação de soluções computacionais traz muitas vantagens em sua utilização. Inclui-se, a eliminação das redundâncias, por esta razão há economia de código e processamento. A manutenção pode ser pontual e mais fácil de se identificar. Facilita a extensibilidade, ou seja, o sistema pode crescer e permanecer útil. Além da reutilização em projetos futuros, assim não precisamos reinventar a roda. Existe maior facilidade de entender e de criar funcionalidades que englobem o sistema existente e possui fácil compreensão.

Listas

São um conjunto sequenciado de dados do mesmo tipo e podem armazenar **n** valores, seu tamanho e tipo são definidos na instância de novo objeto Lista.
As operações possíveis em listas são inserção, exclusão, acesso e distribuição de elementos. Tais operações, dependem do tipo de alocação de memória usada: estática ou dinâmica.

Com capacidade limitada, listas estáticas possuem tamanho pré-definido, os espaços são alocados no momento da compilação e permanecem os mesmos após sua criação.
O acesso aos elementos é feito de forma sequencial:



Ao lado, vemos a representação visual da estrutura de lista estática, nela podemos ver um vetor.

Figura 2: Lista estática (ou vetor) de valores inteiros

Apesar da estrutura de dados exigir a definição do número máximo de elementos na lista, há a possibilidade de que seja redimensionada e o sistema passe a ter uma nova lista de tamanho diferente, armazenando mais informação com novos espaços alocação ou, de forma reduzida, com elementos excluídos da lista, esta ação é executada de forma manual.

Em listas dinâmicas, o espaço de memória é alocado em tempo de execução. A lista cresce à medida em que são inseridos novos elementos e reduzida quando houver remoção de elementos. O acesso encadeado é feito de forma distinta para cada elemento, para acessá-lo, percorre-se todos os antecessores da lista, pois os dados ficam alocados em áreas distintas da memória.

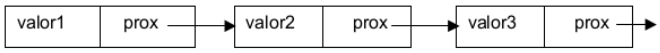


Figura 3: Lista dinâmica
o nó de cada elemento possui um direcionamento para o próximo

A imagem 3, exibe dois campos que uma lista dinâmica possui, o campo de dado, onde o elemento fica alocado, e o apontamento para o próximo campo.

O objetivo principal da estrutura de dados é a organização dos dados na memória ou dispositivo de armazenamento e quando bem aplicado, traz a aplicação melhor desempenho processual. Neste contexto, imaginaremos uma empresa de porte médio, ela possui uma base de dados com mais de mil clientes cadastrados e deseja armazenar em uma aplicação web para melhor atender suas demandas. O desenvolvimento voltado para o melhor desempenho não poderia declarar uma variável para cada cliente, como abaixo:

```
string cliente: “Maria da Silva”;
string cliente2: “José dos Santos”;
string cliente3: “Francisco Almeida”;
```

Cada cliente inserido dessa forma no sistema, teria um processamento alto e de difícil localização. Porém se transformarmos a ‘*string*’ cliente em uma lista do tipo string, podemos alocar os dados e operar de acordo com a posição do elemento na lista, declarando dessa forma:

```
String[] cliente : {“Maria da Silva”, “José dos Santos”, “Francisco Almeida”, ...};
```

De forma simplificada, acima, temos dados do mesmo tipo e interesse armazenados em uma única variável. Em sistemas maiores, a ligação é feita através de banco de dados e tem estrutura similar, mas com capacidade de operação em massa dos elementos.

Herança e Listas

Listas também podem ser declaradas como objetos. Ao serem instanciadas, é possível criar uma classe que possua atributos de uma lista.
Pode receber informações de um banco de dados e alocadas na lista. Essa estrutura, pode aproveitar de todas as vantagens de Herança.
Ao trabalhar em conjunto a estrutura de dados e operar de forma onde cada coisa fique em lugar e tenha o direcionamento adequado para ser acessado.