**ELEMENTOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS**

Para construção do programa descrito nesse trabalho, será utilizado a linguagem de programação Java, e a confecção será baseada no paradigma de programação orientado a objetos. Mas para isso, deve-se primeiro evidenciar o que são e para que servem cada um desses elementos.

Diferente do paradigma estruturado de programação, a orientação a objetos é baseada em recursos de modularização de processos, tentando aproximar esses processos a vida real. Essa modularização do programa é feita a partir de classes, que interagem entre si através da criação de objetos de outras classes (um elemento baseado em generalização), fazendo com que seja possível utilizar seus métodos ou atributos.

Sendo mais descritivo, uma classe é como um molde que possui a função de categorizar um determinado grupo de elementos que serão utilizados durante a execução do sistema. E dentro do paradigma discutido, elas auxiliam muito na visualização de um projeto como um todo, fazendo com que a construção e manutenção do código seja mais dinâmica graças a diminuição da quantidade de linhas utilizadas em cada módulo. Nesse processo de categorização do programa, é possível atribuir a elas algumas ações que os seus elementos podem executar e características, ou qualidades, que são respectivamente os métodos e atributos de uma classe.

Já os objetos servem para conectar esses módulos de acordo com a categoria adotada pela classe, os atributos e os métodos. Com eles é possível executar ações em uma classe que em outra também deveria ser possível, além de criar os elementos que fariam parte desse conjunto designado. Um exemplo simples capaz de demonstrar os conceitos descritos acima, é a imagem abaixo:

**Figura x** – Classes e Objetos

Diagrama

Descrição gerada automaticamente

**Fonte:** <https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos>

Como é possível observar, a classe é uma abstração de uma categorial de um elemento qualquer. Os objetos por sua vez são como os elementos criados propriamente ditos, podendo conter nomes, cores ou ações e qualidades diferentes, desde que elas estejam especificadas na classe.

Dentro do conceito de classe, é possível estabelecer algumas qualidades ou características para o determinado grupo de elementos. Essas características são chamadas de atributos, se parecem muito com variáveis e graças a linguagem Java ser fortemente tipada, é necessário indicar o tipo primitivo ao declará-lo.

Para guardar caracteres, pode ser usava o tipo primitivo char, que armazena em memória um caractere, ou a classe nativa do Java String, que permite armazenamento de diversos caracteres em sequência. Para números existem um pouco mais de opções, sendo elas short para guardar números de -32.768 a 32767, int: -2.147.483.648 a 2.147.483.648 e long: -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Por final, para tipo ponto flutuante (com casas depois da vírgula) podem ser utilizados float para precisão de até 7 casas decimais e double para precisão de 15 casas decimais.

Além dos atributos, é possível criar métodos nas classes, que são como abstrações de ações do elemento físico a se tratar, todavia, são parecidos com funções ou procedimentos da programação estruturada. É possível dois tipos de métodos básicos, o void que apenas é chamado para executar alguma oura função, ou então o método com retorno. Neste citado anteriormente, é necessário adicionar a implementação return e definir um valor correspondente ao tipo primitivo declarado no início do método.

Uma forma de lidar com os atributos e de formatar de forma coerente um programa, é através do encapsulamento. Ao tornar o atributo privado, apenas a classe a qual ele foi criado será capaz de usá-lo e manipulá-lo, no entanto, utilizando os métodos get e set é possível utilizar esses atributos com os métodos utilizando a instância da classe correspondente. O método set recebe um valor do tipo do atributo como parâmetro e faz o atributo receber aquele valor, assim como, o método get serve para pegar e utilizar esse valor atribuído ao atributo. Com isso, é possível definir regras dentro dos métodos para que não sejam alterados ou violados de forma indevida pelo usuário.

Algumas vezes nos programas é necessário alterar um método já declarado em outra classe ou manipulá-lo, para isso, pode ser utilizado o conceito de sobrecarga de métodos. Consiste basicamente em reescrever o método, porém alterando sua assinatura, que são os tipos primitivos dos parâmetros desse método, assim sendo podendo com um mesmo método, executar funções diferentes. Esse é um conceito também de poliformismo, que será especificado mais para frente ao mencionar herança.

Para evidenciar outros conceitos é necessário primeiro estabelecer o conceito de herança entre classes. Uma herança como de forma intuitiva em seu nome diz, é o processo de passagem de algum valor para algum elemento ou indivíduo. Nesse caso é a passagem de todos os atributos e métodos de uma classe, denominada classe-mãe, para outras classes, denominadas classes filhas, através do comando extends. Esse conceito abre muitas possibilidades e funções para um programa, uma delas é a adição de novas classes que sejam semelhantes ou tenham características únicas, porém também as características da classe mãe.

Através da herança, é possível utilizar funções para boas práticas de programação que tornam o sistema muito mais padronizado e entendível para manutenção, reutilização de código e a própria implementação de funções novas, entre eles, o método construtor deve ser citado. O método construtor é responsável pela criação de instâncias de uma classe qualquer em outra, ou seja, o objeto. Sempre que é criado um objeto em uma classe sem criar o método construtor na classe que vai ser instanciada exigindo algum parâmetro na assinatura do método, um construtor vazio é criado pelo próprio Java sem implementação.

Entretanto, criar o método construtor pedindo algum parâmetro que será atribuído a algum atributo da classe ou da classe mãe é uma boa prática de programação, e ao atribuir o parâmetro a classe mãe, é necessário utilizar o comando super. Caso haja mais de um método construtor com o mesmo nome, mas com assinatura diferente, a própria linguagem de programação identificará qual deles deve utilizar com base no parâmetro enviado pelo programa.

Agora com o conhecimento de herança em mente, é possível descrever o que é sobrescrita de um método. Esse conceito é um tipo de polimorfismo que consiste em reescrever um método existente em uma classe mãe e diferente da sobrecarga, mantem o mesmo tipo de parâmetro definido no método, mas cria uma versão daquele método, podendo alterar suas funções e utilizar aquele que o deseja seja na classe filha ou na classe mãe. Este conceito citado é uma clara representação de polimorfismo, que é a utilização de um elemento de formas diferentes.

Finalizando os conceitos principais do paradigma de programação orientado a objetos, é preciso relacionar as funções explicadas com os modificadores de acessos e comportamentos. O modificador public, garante que os elementos de um programa sejam acessados e modificados por qualquer classe em qualquer pacote. Enquanto o protected restringe o acesso desse elemento a apenas acessível através dos pacotes. E o private é responsável por proibir o acesso aos elementos designados de qualquer classe ou pacote. Dessa forma, só é possível acessá-los através dos métodos get e set, que além de padronizar o sistema, adiciona segurança seja da pessoa que está programando, ou de uma situação atípica.

Os modificadores de comportamento como no próprio nome dizem, serve para mudar o comportamento inicial que um determinado elemento do sistema possui. Existem três, são eles, static, final e abstract. Uma classe pode ter apenas um entre final e abstratic, sendo que final torna a classe impossível de ser instanciada, ou seja, criar um objeto em memória, enquanto abstract torna a classe possível de ser instanciada, porém, apenas através de suas classes filhas, obrigando que seja implementado a herança.

Para métodos e atributos, também alguns dos três modificadores pode ser utilizado, sendo que o static torna um método e um atributo mais fáceis de serem acessados e utilizados, possibilitando que sejam manipulados e invocados sem que haja a presença de um objeto da classe da qual correspondem. Já no modificador final, torna tanto o método quanto o atributo impossível de manipular através de uma possível sobrescrita ou sobrecarga, e de alterar valores de um atributo, tornando-o uma constante. Por final para métodos portando o abstract, é obrigatório que os mesmos sejam implementados pelas classes filhas, dessa forma, eles não possuem implementação na superclasse, além disso, só podem existir em classes abstratas.

Para terminar os conceitos principais do paradigma abordado nesse trabalho, um conceito que não tem em todas as linguagens, mas que possui na linguagem utilizada para esse trabalho, é a interface. Existem diversas discussões na web que se trata de se uma interface é uma classe ou algo diferente. Internamente ao codificá-la, se declara como interface, porém, ao salvar e buscar o arquivo criado no diretório guardado, a extensão é a mesma de uma classe. Entretanto, ela possui um comportamento diferente de uma classe normal, bem parecido com o de uma classe abstrata. A diferença entre uma interface e uma classe abstrata é que na interface não se pode declarar atributos e todos os seus métodos devem obrigatoriamente ser implementados em alguma outra classe ao colocar “implements” na declaração dessa classe.