Para suprir um mercado cada vez mais exigente e cheios de necessidades o desenvolvimento de software é extremamente amplo e adaptativo. Afim de resolver os mais diferentes tipos de problemas, existem diversas linguagens de programação, que seguem diferentes paradigmas. Atualmente o padrão que mais tem evoluído e também o mais difundido é a Orientação a Objetos, se destacando principalmente em questões voltadas para segurança e reaproveitamento de código, o que é muito importante no desenvolvimento de qualquer aplicação moderna.

A orientação a objetos tem como foco central o conceito de objeto, e pode representar qualquer coisa que exista no mundo real. A base da Programação Orientada a Objetos (POO) são quatro pilares: abstração, herança, encapsulamento e polimorfismo.

A representação de um objeto em uma linguagem de programação pode não ser uma tarefa simples, contudo, as linguagens de POO possuem suporte ao conceito de classe. A classe é a principal entidade da POO, pois nela conterá todos os atributos e métodos de um determinado objeto. Os atributos são as propriedades e os métodos os comportamentos de um objeto, ao implementar a classe Pessoa, podemos identificar alguns atributos como: nome, idade, sexo, entre outros e alguns métodos como: andar, falar, correr, entre outros. O processo de mapear todos os atributos e métodos necessários para uma classe é chamado de Abstração que é o nosso primeiro pilar. A classe não é o objeto propriamente dito, mas sim, um template para a construção de qualquer objeto, por exemplo, Maria e Pedro são instâncias da classe Pessoa. Uma outra característica da classe é sua identidade, que deve ser única dentro do sistema para que não haja conflito. A nomenclatura da classe deve possuir uma semântica dentro do contexto da aplicação e por convenção o nome da classe deve iniciar sempre com a inicial em maiúscula (Pessoa) e se o nome for composto, as iniciais de cada palavra em maiúscula (PessoaFisica, PessoaJuridica) e não usar ifém, underline ou espaço para separar as palavras, elas devem está unidas. Mesmo isso sendo uma convenção algumas linguagens de programação podem gerar um erro caso essa não seja obedecida. Não existe um limite da quantidade de atributos e métodos, mas a classe deve manter o principio da responsabilidade, assim, qualquer objeto criado por esta classe terá os mesmos atributos e comportamentos.

Cada linguagem de programação pode implementar uma classe de forma diferente, contudo, existem algumas convenções que independem da linguagem devem ser seguidas e a principal é o identificado de classe

Outro pilar é o encapsulamento e consiste em esconder a estrutura de dados utilizada e somente prover uma interface de acesso bem definida. Se trata de um dos elementos que adicionam segurança à aplicação ao encapsular propriedades, criando uma espécie de caixa preta. A interface com cada módulo é definida de forma a revelar o mínimo possível sobre o seu funcionamento interno. É importante ressaltar que objetos se comunicam por meio de mensagens, assim a interface de acesso fornece o conjunto de todas as mensagens a que ele pode responder e é o próprio objeto que seleciona o método pelo qual ele irá reagir à mensagem. Ao encapsular dados de uma classe algumas propriedades de uma determinada classe devem ser privadas, e quando outra classe necessitar verificar o estado de uma propriedade, fará utilizando métodos de acesso implementados. A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados getters e setters, que irão retornar e setar o valor da propriedade, respectivamente. Essa atitude evita o acesso direto a propriedade do objeto, adicionando uma camada de segurança à aplicação. Outra característica que é especifica de algumas linguagens de programação, são os modificadores de acesso, público (public), privado (private) e protegido (protect). As propriedades públicas podem ser acessadas livremente por qualquer outra classe. Propriedades privadas são visíveis apenas dentro da classe que as implementa e as protegidas são semelhantes as privadas com a diferença que podem ser visíveis para seus herdeiros.

A herança é outro pilar importante da POO que permite otimizar a produção da aplicação em tempo e linhas de código, e consiste em reutilizar código já inscrito, de forma que as classes derivadas devem se comportar como as superclasses existindo uma hierarquia de Generalização/Especialização entre as classes. O significado de herança tem o mesmo significado para o mundo real, ao imaginar uma família, o filho herda características dos pais, e por sua vez vão passar para seus filhos. Na POO uma classe pode herdar informações de uma outra classe maior chamada de superclasse ou classe pai. De forma mais precisa, a subclasse ou classe filha herda os atributos e métodos que são compartilhados em modo privado (private). Os métodos de uma subclasse não acessam membros private diretamente de sua superclasse. E para alterar o estado de variáveis de instância private da superclasse é necessário utilizar um método não-privado fornecido pela superclasse e herdados pela subclasse. A classe Pessoa seria uma classe genérica de forma a representar uma generalização das classes inferiores, de certo que as classes PessoaFisica e PessoaJuridica seriam especializações da superclasse, além disso é possível acrescentar novas características, permitindo um comportamento próprio. A herança não está limitada a apenas um nível, a hierarquia de classe pode ser tão profunda quanto necessário e quanto mais alto seu nível mais especializada é o seu comportamento. Para melhor compor um objeto, é comum introduzir classes que não estão presentes no mundo real, sendo apenas estruturas conceituais, essas são classes abstratas e costumam ser apenas um repositório de atributos. É importante salientar que o uso excessivo de herança aumenta o acoplamento (forte acoplamento) entre as classes, isto é, o quanto uma classe depende de outra. Estudos demonstram que quanto menor for o nível de acoplamento (fraco acoplamento) mais fácil será a sua manutenção e o reuso se tornando importante para melhorar a qualidade do software. Para algumas linguagens de POO a herança não está limitada a uma única superclasse, podendo herdar características de diversas classes, introduzindo o conceito de heranças múltiplas.

Na natureza tudo se transforma, ja diria Lavoisier, com os objetos algo semelhante acontece. Quando uma classe herda informações de outra ele absorve seus comportamentos, contudo alguns desses métodos são reescrito por cada objeto ter uma característica diferente. Vejamos a classe Veiculo, nela existe uma propriedade que diz que objetos do tipo veículos possuem uma ignição de partida e essa propriedade é replicada aos herdeiros, porem, na classe Moto podemos ter um método de ignição elétrica ou via pedal, e a classe Carro também podem ter uma partida elétrica ou manual usando uma chave, de forma que cada objeto herda conjuntamente a propriedade ignição da superclasse, ambos objetos reescrevem esse comportamento de formas diferente. Logo, o conceito de *polimorfismo* consiste na alteração do funcionamento interno de um método herdado de uma superclasse.

De modo geral, orientação a objeto é um paradigma que está no mercado de desenvolvimento de software desde a década de 80, apesar disse as linguagens de programação evoluíram em sua maioria para atuar onde uma determinada linguagem pode falhar ou melhor, tratar de forma particular um determinado tipo de problema. Sendo assim, alguns conceitos da orientação a objetos pode ser interpretado de forma distinta por uma ou outra linguagem. O mais importante a partir desse ponto de vista é o desenvolvedor conhecer com bastante propriedade o escopo do problema a ser resolvido e optar pela linguagem que mas se adeque a resolução do problema, assim você não precisa declarar amores a linguagem x ou y, mas sim utilizar seja ela qual for para entregar uma solução que supra as necessidades dos seus clientes ou usuários.