O Paradigma de Programação Orientada à Objetos é fundamentado principalmente na utilização de estruturas pré-definidas (moldes) de artefatos existentes ou não no mundo real, sendo essas nomeadas de Classes. Utilizando-se de uma analogia, uma Classe seria a planta de um casa, enquanto a casa (físico) em si seria uma Instância ou um Objeto dessa Classe.

Adentrando na estrutura em destaque, essa possui atributos: variáveis que representam suas características. Enquanto o que pode realizar, ou seja, suas ações, são conhecidas por métodos: de modo geral são funções, mas devido ao contexto recebem outra nomeação.

Prosseguindo no assunto de métodos, em Java é permitido que se passe mais de um método com o mesmo nome e que receba diferentes tipos de parâmetros; e de acordo com os argumentos passados que irá dizer qual rotina será executada. Isso é conhecido como Sobrecarga de Métodos, que pode ser útil para prosseguir de diferentes maneiras a partir de problemas de contextos congruentes.

Toda Classe possui um método especial chamado de *Construtor*, mesmo que vazio. Tal função de Classe deve possuir o mesmo nome da Classe, não ter retorno e é executada no momento do instanciamento. Ademais, tem como responsabilidade ditar as regras para a criação de um Objeto, por exemplo: quais atributos são obrigatórios e que devem ser recebidos para um Objeto existir.

Outra base do POO é o conceito de Herança. Nessa ótica, dizer que uma Classe B herda de A ou que A é uma SuperClasse de B ou ainda que B é uma SubClasse de A, significa falar que B tem todos os atributos e métodos de A, além dos seus próprios, o que pode ser útil para o desenvolvimento de códigos mais eficientes e de melhor manutenção. Também vale ser mencionada a existência de Classes Abstratas, isto é, Classes que não podem ser instanciadas diretamente, mas servem como base/podem ser herdadas por outras Classes.

Agora que foi conhecido o contexto de Super e Sub Classe, pode-se trazer o assunto de Sobrescrita de Métodos, como o nome sugere, seria a capacidade de uma SubClasse conseguir alterar a implementação de um método herdado e assim, conseguir personalizar e adaptar às suas necessidades próprias.

Ainda no contexto de Herança, há situações em que uma Classe consegue se modificar ou se adaptar, por exemplo: de forma didática, se existir duas Classes chamadas de *Carro* e *Moto* e essas herdam da Classe *Veículo*; Na existência de outra Classe *Mecânica* que tem o método *alterarCorVeiculo*, sendo exigido passar

um *Veículo* como parâmetro, nessa situação, tanto uma instância de *Carro* como de *Moto* podem ser utilizadas para esse método. Para estas ocorrências na qual objetos diferentes podem ser tratados de forma semelhante dá-se o nome de Polimorfismo.

Como os atributos e os métodos são de vital importância para o funcionamento de um Objeto, existem formas de alterar seu modo de interação com o resto do ambiente. Uma delas é a Visibilidade, quando um método ou um atributo é passado como *public*, eles podem ser acessados por todas as Classes, independentemente do pacote que estão localizados. Todavia, na utilização de *protected*, a acessibilidade é permitida dentro da Classe em questão e de outras que herdem. Já para o *private*, o acesso apenas é possível dentro da Classe "principal", sendo que para no caso dos atributos poderem ser manipulados externamente existe a necessidade de serem incrementados métodos específicos para obter (get) e/ou alterar (set) seu conteúdo, esses métodos seguem um padrão de nome *get<nome do atributo>* ou *set<nome do atributo>*.

Uma caminho para proporcionar segurança para os elementos de uma Classe é por meio do Encapsulamento, nele existem princípios que garantem a confidencialidade e a integridade de atributos e métodos. Também é válido destacar que o Encapsulamento utiliza-se de modificadores de Visibilidade e o mal entendimento da aplicação desses poderá resultar na quebra das bases de Encapsular.

Referente a algumas Palavras Reservadas em Java, é pertinente o entendimento de *this, super* e *final*. A primeira serve para se referir a própria Classe que está localizada, sendo sua chamada: *this*, *this.<atributo>*, ou *this.<método>*. A segunda, funciona de forma análoga a *this*, no entanto, na perspectiva de SuperClasse. Em conclusão, *final* é utilizado para declarar que, no contexto de POO, uma Classe, um Método ou um Atributo não podem ser modificados, para Métodos e Atributos significa que são constantes, para Classes, além de constante, que não pode ser herdada.

Sobre a Relação entre Objetos, quando uma Instância "tem" ou possui outra, significa que um dos atributos da Classe "principal" é do tipo de outra Classe. Já o laço de "usar" vem em casos na qual um Objeto usa de outro para efetuar uma ação, geralmente isso acontece nos métodos. Por fim, outra conexão entre Objetos é de "ser", que acontece quando uma classe herda a outra.