Em Java orientado a objetos existe uma série de termos e conceitos importantes para poder utilizar a linguagem corretamente, para começar, os **objetos** são muito utilizados na programação orientada a objetos, eles são como uma maneira de de representar entidades do mundo real dentro do código, além de simplificar a lógica de programação, dividindo as responsabilidades entre vários objetos distintos. Os objetos são criados a partir da palavra “new” e podem ser definidos por uma classe. A **classe** é uma estrutura fundamental que define um tipo de objeto, uma classe pode ser pensada como um modelo/plano para criar objetos que vão possuir atributos e métodos específicos. Os **métodos e atributos** são definidos dentro de uma classe, um atributo é uma variável definida dentro de uma classe que define características ou propriedades de um objetos e podem ser acessados através de métodos, um método, por sua vez, é uma função definida dentro de uma classe que pode ser invocada por um objeto daquela classe. Os métodos e atributos podem ter diferentes tipos de visibilidades, como **public, private e protected**. Que está diretamente ligado com o conceito de **encapsulamento** que refere-se a prática de esconder o estado interno de um objeto, ou seja, o acesso aos atributos de um objeto é limitado e controlado através de métodos específicos.

No Java existe o conceito de **herança** que é implementada usando a palavra-chave “extends”, a herança é um conceito que permite que uma classe herde atributos e métodos de outra classe. A classe que herda é chamada de subclasse e a que é herdada é chamada de superclasse, um conceito que está atrelado com herança é o conceito de **polimorfismo** que é a capacidade de um objeto de ser tratado como se fosse de vários tipos diferentes ele possui dois mecanismos principais: sobrecarga de método e sobrescrita de método, a **sobrecarga de método** é quando uma classe tem dois ou mais métodos com o mesmo nome, mas argumentos diferentes. Isso permite que a classe lide com diferente tipos de entrada e execute a ação apropriada de acordo, a **sobrescrita de método**, por sua vez é quando uma classe filha substitui um método da classe mãe com sua própria implementação, isso permite que a classe filho herde métodos da classe mãe e personalize para atender às suas necessidades.

Em paralelo existe a **classe abstrata** que é uma classe que não pode ser instanciada diretamente, mas é projetada para ser estendida por outras classes que fornecem uma implementação mais especifica, uma classe abstrata é declarada usando a palavra-chave “abstract” e podem ter métodos abstratos, que são métodos que não possuem implementação e são definidos apenas pela sua assinatura, ela é projetada para ser uma classe base para outras classes que estendem dela. As classes abstratas assim como as classes normais possuem um **construtor** que é um método especial que é usado para inicializar um objeto assim que ele é criado. O construtor é chamado automaticamente na criação do objeto usando a palavra-chave “new” e é usado para configurar os valores iniciais dos campos do objeto. Outros métodos bastante conhecidos são os métodos **Getters** e **Setters**, que são métodos usados para acessar e modificar valores dos atributos de um

objeto de uma classe. Os métodos Getters são geralmente definidos com o prefixo "get" seguido pelo nome do campo e retornam o valor atual do campo, eles são usados para permitir que outras classes leiam o valor do campo sem ter acesso direto ao campo em si, eles geralmente são usados para ler valores de campos encapsulados. Já os métodos Setters são geralmente definidos com o prefixo "set" seguido pelo nome do campo, e permitem que outras classes alterem o valor de um campo específico. Eles são usados para permitir que outras classes modifiquem o valor de um campo privado sem ter acesso direto ao campo em si, eles também são usados para mexer com campos encapsulados. Esses métodos são uma parte importante do encapsulamento pois permitem que as variáveis privadas de uma classe sejam acessadas e modificadas de forma controlada e segura. Eles também fornecem uma maneira de validar ou manipular os valores passados para uma variável antes de serem armazenados em um campo.

Em Java, existem algumas palavras reservadas que têm significados específicos e são usadas para implementar recursos importantes da linguagem. Algumas dessas palavras reservadas são: "super", "this" e "final". A palavra reservada "**super**" é usada em Java para acessar um membro de uma classe pai. Quando uma classe herda de outra classe, a palavra reservada "super" pode ser usada para acessar os membros da classe pai que foram ocultados ou sobrescritos na classe filha. Já a palavra reservada "**this**" é usada para se referir ao objeto atual da classe em que está sendo usado, por exemplo se no método construtor a gente receber uma variavel “id” e já existe um atributo chamado “id” dentro da classe, então para receber esse valor, vai ser necessário utilizar a palavra this para indicar que o atributo id daquela classe vai receber o valor da váriavel, exemplo: this.id = id. A palavra reservada "**final**" é usada em Java para indicar que um campo, método ou classe é "final". Um campo "final" não pode ser alterado depois que é inicializado, um método "final" não pode ser sobrescrito por uma subclasse é uma classe "final" não pode ser estendida por uma subclasse. O uso de "final" é comum em Java para indicar que uma entidade é imutável ou não deve ser modificada em tempo de execução.

Em programação orientada a objetos, é comum que as classes se relacionam entre si, podendo se relacionar de três maneiras principais: "ter", "usar" e "ser". A relação "**ter**" é uma associação entre classes, em que uma classe possui uma referência a outra classe. Essa relação indica que uma classe é proprietária de outra classe e é responsável por gerenciá-la. A relação "**usar**" é uma composição entre classes, em que uma classe é composta por outra classe. Nesse caso, a classe composta é uma parte da classe que a utiliza. Essa relação indica que uma classe é dependente de outra classe. Por fim, a relação "**ser**" é uma herança entre classes, em que uma classe herda as propriedades e métodos de outra classe. Essa relação indica que uma classe é uma extensão ou especialização de outra classe.