FIAP ON

JAVA FUNDAMENTOS

ORIENTAÇÃO A OBJETOS

THIAGO YAMAMOTO

1

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Nos últimos anos, o mundo presenciou uma ampla transformação tecnológica. Sua utilização em diversas áreas é uma realidade e seus impactos são positivos, trazendo progresso e agilidade por onde passa.

Pense em uma loja, um banco financeiro, um hospital, sua conta de telefone ou na sua rede social favorita. Todos eles precisam de um sistema para serem gerenciados. É aí que entra o Desenvolvedor de Sistemas.

Esse profissional de TI, tão requisitado, é fundamental na construção de sistemas e deve dominar conceitos básicos, entre eles: a lógica e a linguagem de programação.

O mercado atualmente dispõe de várias linguagens de programação para a construção de softwares. Novas linguagens podem surgir e as “antigas” evoluem, com novas versões.

A escolha de uma linguagem para o desenvolvimento de um sistema deve levar em consideração diversas variáveis estratégicas, como: maturidade da plataforma, custo, velocidade de desenvolvimento e curva de aprendizagem.

Java vem se tornando uma plataforma de desenvolvimento das mais utilizadas e aceitas no mercado. Isso se deve a algumas características, como: portabilidade, simplicidade e entre outros que iremos elencar no decorrer do curso.

Aprender a linguagem Java não limita o desenvolvedor a esta plataforma. Os conceitos de Orientação a Objetos: Herança, Polimorfismo, Encapsulamento, Modularização e Abstração são aplicados a qualquer tipo de linguagem Orientada a Objetos (C#, VB.NET, PHP etc.) e, a lógica de programação, estruturas de controle, variáveis, por exemplo, são comuns a todas elas.

1.1 Linguagem de Programação

Uma linguagem de programação pode ser definida como sendo um conjunto limitado de instruções (vocabulário), associado a um conjunto de regras (sintaxe) que define como as instruções podem ser associadas, ou seja, como se podem compor os programas para a resolução de um determinado problema.

A programação em computadores teve início com o objetivo de resolver cálculos matemáticos, depois passou a ter uso nas empresas em meados dos anos 1970, com objetivo da automatização de processos manuais. O formato que dominava a programação dos sistemas na época era chamado de programação linear e estruturada, que tem como base o controle: sequência, condição e repetição, e a subprogramação (ou modularização) utilizando sub-rotinas e funções.

Nesta categoria encaixam-se as chamadas linguagens de programação de baixo nível, surgidas em meados dos anos 1960 como Fortran, Cobol, Pascal, C.

As linguagens concebidas nesse período resultam da necessidade da produção de código de programa de forma clara, aparecendo o conceito de estruturação do código. O período compreendido entre as décadas de 1960 e 1980 foi bastante produtivo no que diz respeito ao surgimento de linguagens de programação, o que permitiu o aparecimento de uma grande quantidade de linguagens.

Com o aumento na produção de sistemas e com a expressa necessidade de reaproveitamento de códigos, este processo antes inicialmente tão positivo, passou a se tornar repetitivo e trabalhoso, pois toda e qualquer atualização no sistema deveria ser realizada em todas as partes que eram replicadas.

Mediante esta situação que, além do desgaste técnico, causava baixa produtividade no desenvolvimento e peso financeiro às empresas, novas técnicas surgiram para resolver estes problemas. Desponta o modelo de objetos, baseado nos moldes já milenares da vida do ser humano.

Trazer para a programação os conceitos de objetos da vida real e a criação de moldes para esses objetos, favoreceu toda uma nova linha de sistemas otimizados, reaproveitáveis e de fácil atualização.

1.2 Introdução Java

A primeira versão da linguagem Java surgiu em 1995, criada por um time de desenvolvedores da empresa Sun Microsystem, liderado por James Gosling. A ideia inicial era desenvolver uma linguagem para controlar pequenos dispositivos, como televisores, videocassetes e aparelhos de TV a cabo. A ideia não deu certo, mas com

o advento da internet, a plataforma foi adotada, se expandiu e evoluiu, e atualmente é uma referência no mercado de desenvolvimento de sistemas.

Java se tornou popular pelo uso na internet e hoje roda em muitos equipamentos e dispositivos: notebooks, celulares, videogames, cartões inteligentes etc.

Podemos elencar várias características-chave para a plataforma ter alcançado tanto sucesso:

- **Simples:** a sintaxe do Java é uma versão limpa da sintaxe das linguagens da época, como C++. Não há a necessidade de arquivos de cabeçalho ou trabalhar com ponteiros (alocar memória da máquina para armazenar informações).
- **Robusto:** Java foi concebido para desenvolver programas confiáveis, em vários aspectos. Existe uma verificação preliminar de possíveis problemas, que em outras linguagens, só seriam descobertos em tempo de execução.
- **Seguro:** Java é utilizado em ambientes de rede/distribuído. Desse modo, muito se trabalhou para a segurança, deixando a plataforma livres de vírus e adulterações.
- **Alto desempenho:** o código Java é convertido em bytecodes (veremos com mais detalhes), esses bytecodes são interpretados em um ambiente de execução do Java para executá-los. Se for necessário mais desempenho, esse ambiente de execução transforma os bytecodes em código de máquina nativo para a CPU específica, ganhando assim desempenho.

Além desses, podemos citar duas outras características que foram determinantes para o sucesso: portabilidade e orientação a objetos.

1.3 Portabilidade

Entende-se por portabilidade a capacidade de ser utilizado em qualquer plataforma, neste caso – sistema operacional (Windows, Linux, Mac OS.) e hardware. Ou seja, Java é utilizado independente de plataforma.

Um programa Java gerado no ambiente Windows pode, facilmente, ser executado em Linux, sem nenhuma alteração no código.

Quando escrevemos código utilizando uma linguagem de programação é necessário a conversão desse código para um outro código, de modo que o computador seja capaz de executá-lo. Esse processo possui o nome de **compilação,** cujo significado é transformar o código fonte, que é mais fácil para os desenvolvedores, em código de máquina, que é utilizado pelo computador.

Linguagens como C e Pascal são compiladas para um determinado sistema operacional e arquitetura de hardware, ou seja, após a compilação, o código executável (binário) roda somente para um tipo de sistema operacional. Se for necessário executar em outro ambiente, será preciso compilar o programa especificamente para esse ambiente.

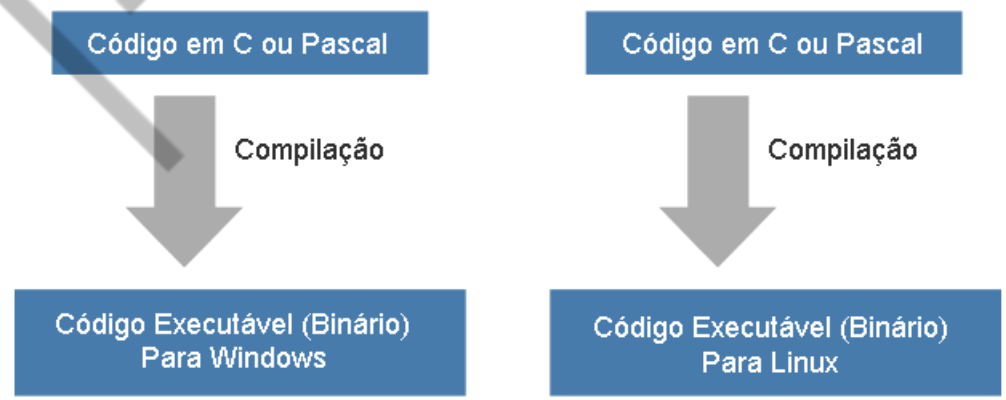

Figura 1.1 – Compilação de código Java para um SO específico
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

Em algumas situações haverá necessidade de realizar ajustes no código, para o perfeito funcionamento em cada ambiente. Dessa forma, temos um código executável para cada sistema operacional.

A linguagem Java é diferente. Existe uma **máquina virtual**, conhecida como **JVM** (*Java Virtual Machine*), que é capaz de interpretar (executar) os arquivos Java compilados.

A linguagem Java é **Compilada** e **Interpretada**. Primeiramente, os arquivos de código fonte Java com extensão ".java" são compilados para *bytecodes,* também conhecidos como arquivos de extensão ".class".

Após esse processo, os *bytecodes* são interpretados pela JVM, iniciando a execução do software. Para cada plataforma (sistema operacional + hardware) existe uma Máquina Virtual Java, tornando as aplicações Java portáveis.

A imagem abaixo ilustra um código Java sendo compilado, que pode ser executado em diferentes plataformas em suas respectivas JVMs.

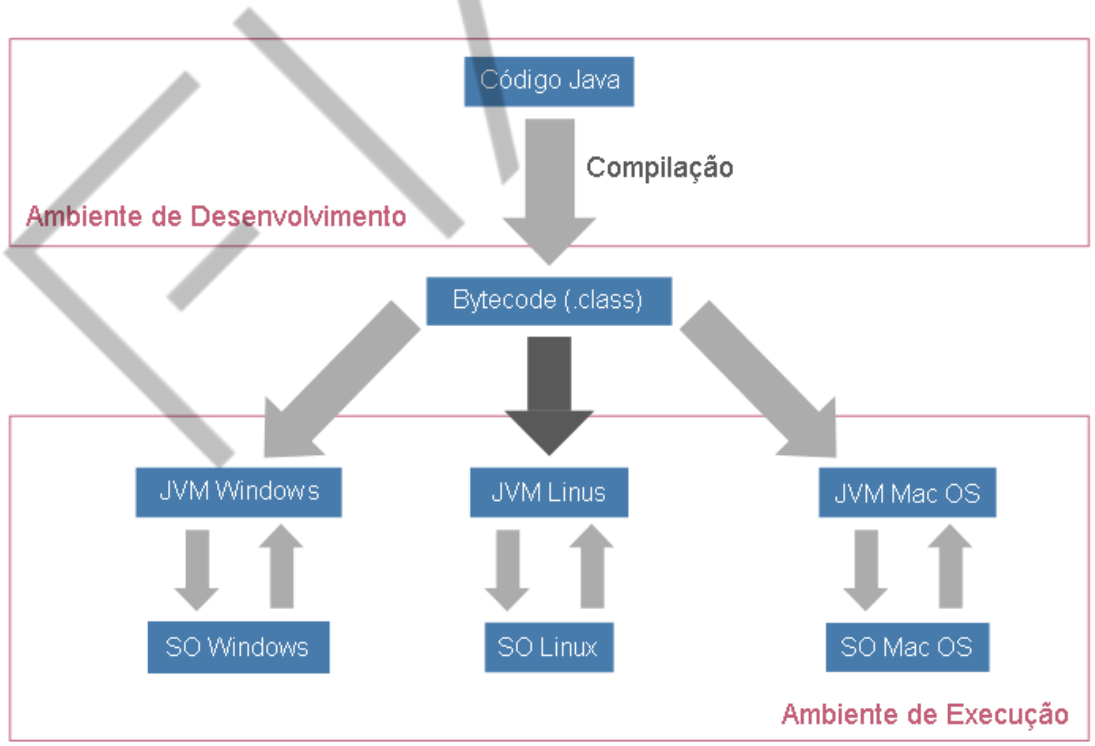

Figura 1.2 – Funcionamento do processo de compilação, distribuição e execução de código Java
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

Portanto, uma vez compilado, podemos executar o programa independentemente da plataforma utilizada. Por exemplo, se estamos utilizando um ambiente com o sistema operacional Linux e o hardware da família x86 para desenvolver o nosso programa, uma vez compilado, o aplicativo pode ser executado em um ambiente Windows ou Mac OS, sem a necessidade de recompilar.

1.4 Orientação a objetos

A programação orientada a objetos é uma técnica de programação que focaliza os dados e interfaces com esses objetos. A essência da programação orientada a objetos consiste em tratar os dados e os procedimentos que atuam sobre os dados como um único objeto – uma entidade independente com uma identidade e certas características próprias.

Para melhor representar a diferença, façamos a analogia com uma fábrica de carros. Para fabricar carros, é necessário que seja desenhado um molde com as propriedades que os carros fabricados terão, tais como: modelo, potência, cor, entre outros. A partir de um único molde, é possível fabricar um, dois ou um milhão de carros, cada qual com sua potência, cor e modelo. Portanto, um mesmo molde é utilizado para a criação de vários objetos com caraterísticas diferentes.

Figura 1.3 – Molde para construção de carros
Fonte: Banco de imagens Shutterstock (2017).

Esta analogia é aplicada na criação de sistemas através da Programação Orientada a Objetos, cujos moldes são criados pensando no mundo real e são chamados de Classes. Em tempo de execução (fabricação), os objetos são gerados cada qual com suas informações. Caso uma atualização seja necessária, a Classe é alterada e a mudança é refletida para todos os sistemas que utilizam este molde e, consequentemente, a todos os objetos gerados também.

Um objeto, como já descrito, é gerado a partir de um molde ou classe, seguindo os princípios do mundo real. Um objeto representa uma entidade que pode ser física, conceitual ou de software:

- Física: quando representa um modelo físico (como um caminhão, óculos, prédio, entre outros).
- Conceitual: quando representa formas abstratas, não palpáveis (como a matemática, o pensamento, o sentimento, entre outros).

- De software: quando representam sistemas (como um usuário, e-mail, acesso etc.).

Um objeto é uma entidade com fronteira e identidade bem definidas que encapsulam o estado e comportamento.

O **estado** é representado pelos **atributos e relacionamentos**. É a condição ou situação durante a vida de um objeto, que satisfaz alguma condição, realiza alguma atividade ou aguarda algum evento. Um estado normalmente é alterado ao longo de um tempo.

Portanto, o **atributo** representa as informações que o objeto pode ter, como exemplificado na figura abaixo, os atributos são: nome, matrícula, admissão, cargo, disciplina e carga horária.

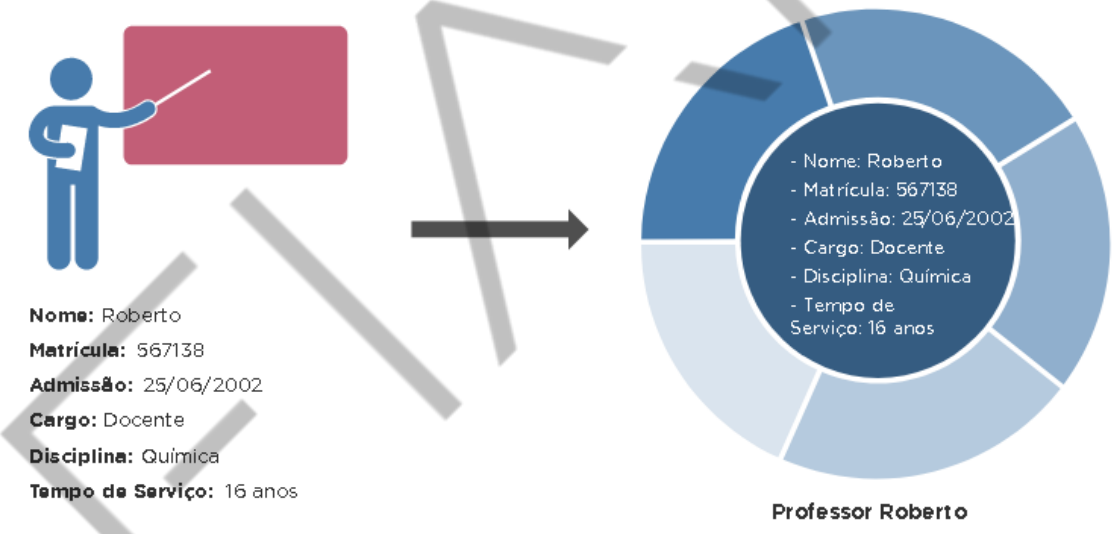

Figura 1.4 – Atributos da classe Professor
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

O **comportamento** é representado pelas operações, métodos e máquinas de estado. Ele determina como o objeto age ou reage a uma requisição de outro objeto. É representado pelas operações que ele pode realizar, ou seja, conforme mostra a figura abaixo – todos os comportamentos que o Professor Roberto pode fazer representam os métodos a ele designado.

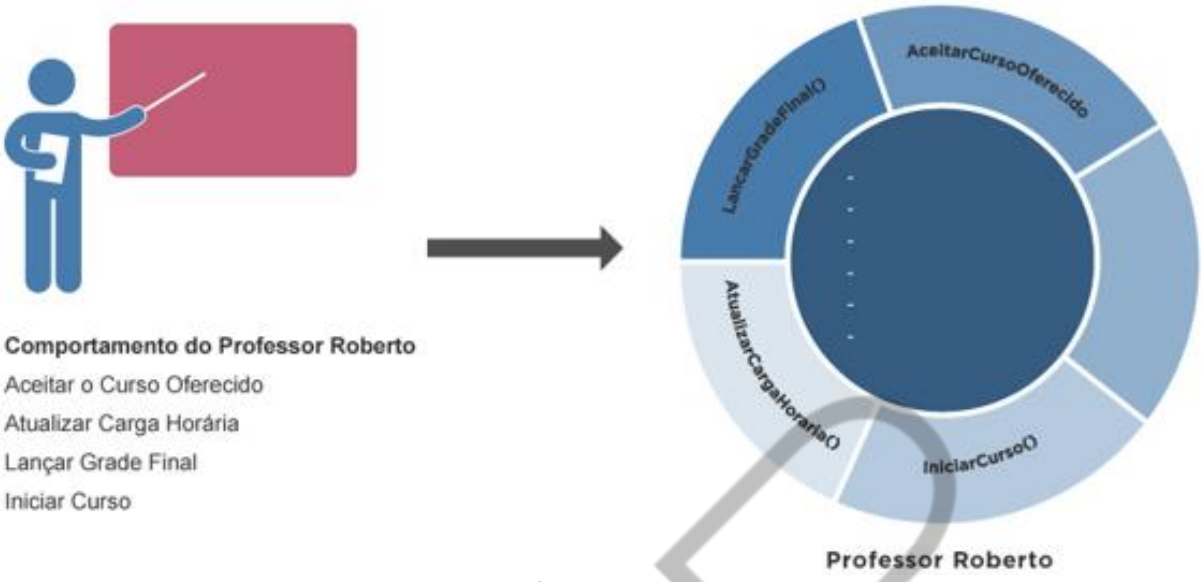

Figura 1.5 – Métodos da classe Professor
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

Portanto, a programação orientada a objetos é uma sequência de instruções enviadas ao computador que utiliza e manipula objetos em seu funcionamento.

Prática:

- Pense em dois objetos do mundo real.
- Para cada objeto:
- Identifique quatro atributos (propriedades).
- Descreva três comportamentos (métodos).

1.5 Princípios de Orientação a Objetos

O alicerce da Programação Orientada a Objetos é baseado em quatro princípios, os quais devem ser pensados e refletidos nos processos do desenvolvimento de um sistema:

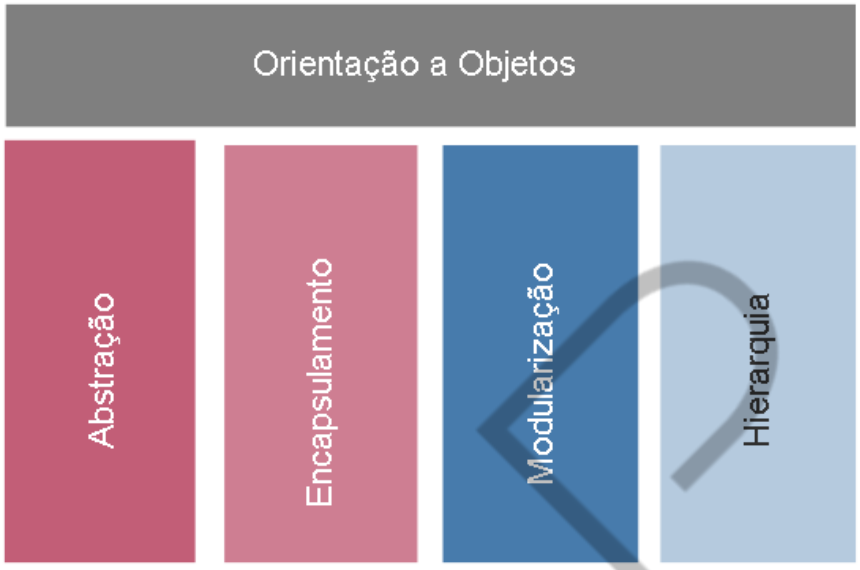

Figura 1.6 – Princípios da programação orientada a objetos
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

- **Abstração:** entendimento e análise das necessidades do sistema, abstraindo o do mundo real.
- Exemplos: para um sistema de gerenciamento acadêmico de uma faculdade, mais especificamente para a funcionalidade de cadastro de alunos, os moldes (classes) devem ser pensados com base nas características reais de um aluno (registro, nome, endereço, curso, data de nascimento etc.) e que sejam relevantes para o sistema. Por exemplo: informações como altura e peso podem ser irrelevantes para o sistema.
- **Encapsulamento**: programar em partes, o mais isolado possível, encapsulando-as. O objetivo é tornar o software mais flexível, fácil de modificar e evoluir. As classes devem proteger as suas informações e comportamentos, ou seja, devem possuir proteção no acesso a seus atributos e métodos.
- Pensando no mundo real, os objetos também possuem essa proteção.

- Exemplo: quando um cliente compra um micro-ondas e deseja utilizá-lo, basta acionar os botões para programar um tempo (inserindo informações aos seus atributos) e pressionar o botão para iniciar o aquecimento (invocando um comportamento). Esses botões irão acionar os circuitos elétricos e as placas internas, sem que o cliente tenha a necessidade de conhecimento do que ocorre dentro da “CÁPSULA” do micro-ondas.
- As classes devem ter a mesma proteção, permitindo que outras classes somente acessem as suas funcionalidades, sem a necessidade de conhecer a sua lógica interna. Dessa forma, a classe esconde a implementação de classes clientes e faz com que estes interajam somente através das interfaces disponíveis (Botões do micro-ondas).
- **Hierarquia**: na programação orientada a objetos existe herança, assim como no mundo real. Por exemplo: um filho recebeu de seu pai a cor do olho azul e seu pai herdou de seu avô essa característica.

No mundo da programação é o mecanismo pelo qual uma classe pode estender (herdar) de outra classe para receber seus comportamentos (métodos) e variáveis (atributos).

Exemplo: a classe mamífero é pai da classe humano, ou seja, um humano é um mamífero e recebe os atributos e métodos da classe pai (mamífero).

A Herança é um dos mecanismos fundamentais para as linguagens que suportam o paradigma OO (Orientação a Objetos). É um mecanismo que possibilita a criação de novas classes a partir de uma já existente. É utilizada como forma de reutilizar os atributos e métodos de classes já definidas, permitindo assim derivar uma nova classe mais especializada a partir de outra classe genérica existente.

Aplicar herança sempre envolve basicamente dois elementos: uma superclasse (classe pai) e uma subclasse (classe filha).

Superclasse é também conhecida como classe ancestral ou classe pai. Apresenta as características genéricas de um conjunto de objetos.

Subclasse é também conhecida como classe descendente ou classe filha. Ela estende a superclasse para incluir suas características. A imagem a seguir ilustra a herança:

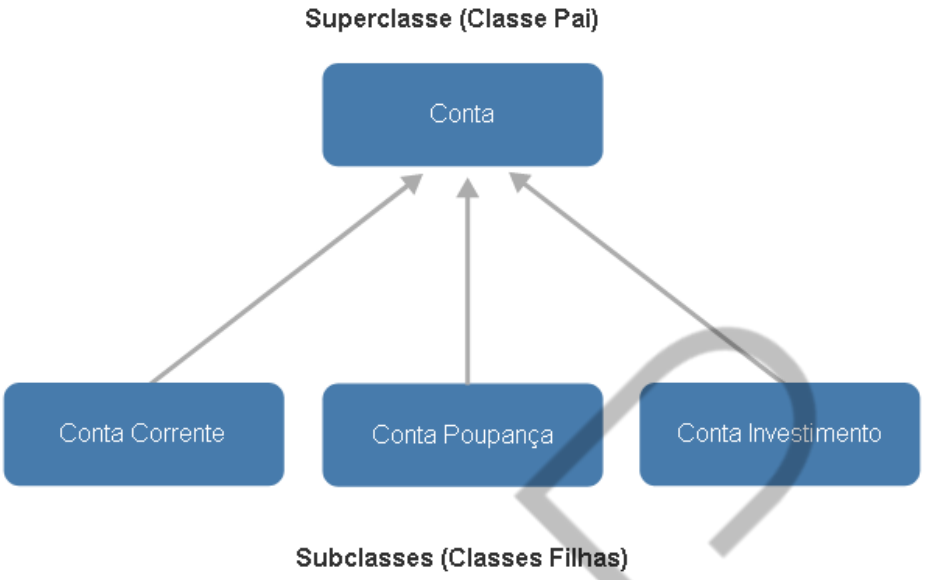

Figura 1.7 – Herança entre classes
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017).

A organização das classes deve ser realizada hierarquicamente, e duas perguntas são básicas para esta organização: *IS-A* (É-UMA) e *HAS-A* (TEM-UMA). É necessário perguntar se uma classe É-UMA parte hierárquica de outra, ou se contém outra classe.

Exemplo: supondo as classes **Aluno**, **Pessoa** e **Disciplina**.

- Aluno **É-UMA** Pessoa? Aluno **TEM-UMA** Pessoa?
  - A resposta para a primeira pergunta é **SIM**, portanto o Aluno tem uma ligação hierárquica com Pessoa (Herança).
- Aluno **É-UMA** Disciplina? Aluno **TEM-UMA** Disciplina?
- A reposta da segunda pergunta é **SIM**, portanto o Aluno tem uma ligação direta com Disciplina, ou seja, a classe Aluno pode possuir um atributo para armazenar as Disciplinas ligadas a ele.

Modularização: é o processo de dividir um todo em partes bem definidas, que podem ser construídas e examinadas separadamente e que consigam interagir entre si. Assim como no mundo real, a programação orientada a objetos permite a criação

de componentes modulares, que podem ser reutilizados para diversos sistemas distintos.

Por exemplo: a classe Aluno, se for criada seguindo princípios citados anteriormente, pode ser utilizada para o sistema de cadastro de alunos, para o sistema financeiro da faculdade, lançamento de notas, consulta ao boletim, entre outros.

Veremos no decorrer do curso como implementar em código Java cada um dos princípios de orientação a objetos apresentados aqui.

REFERÊNCIAS

BARNES, David J. **Programação Orientada a Objetos com Java**: Uma introdução Prática Utilizando Blue J. São Paulo: Pearson, 2004.

COLHO, Alex. **Java com Orientação a Objetos.** Rio de Janeiro: Ciência Moderna, 2012.

HORSTMANN, Cay; CORNELL, Gary. **Core Java**. Volume I - Fundamentos. 8.ed. São Paulo: Pearson 2009.

SIERRA, Kathy; BATES, Bert. **Use a cabeça! Java.** Rio de Janeiro: Alta Books, 2010.