

Separando as Coisas: Criando Serviços

Apresentação



Fonte: <https://goo.gl/toQzMH>

Nesta aula, você aprenderá a separar e organizar sua codificação de forma que você obtenha um maior grau de reaproveitamento daquilo que já foi implementado. Você será apresentado aos métodos no Java e aprenderá a criar classes de dados e de serviços.

Esta aula é o primeiro passo rumo à orientação a objetos. Um bom programador em linguagens orientadas a objetos tem que ser capaz de criar soluções focadas em serviços (métodos) coesos. Afinal de contas, objetos fornecem serviços e operam sobre os dados. Separar os serviços de forma organizada para potencializar o reaproveitamento, esse é o foco.

Siga as orientações, pegue um pouco mais de café e vá em frente.

Conteúdo

Métodos

Nas disciplinas anteriores, você conheceu as funções e procedimentos. Trechos de códigos que eram rotineiros e que poderiam aparecer várias vezes durante a codificação, você transformava em sub-rotinas, ou seja, funções ou procedimentos e os chamava sempre que possível.

Para Refletir

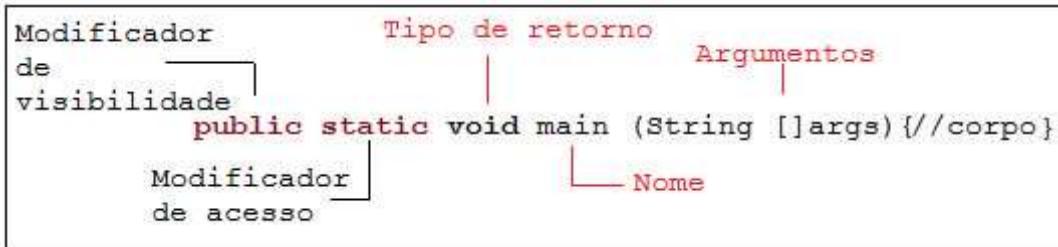
No processo de construção de software, as manutenções adaptativas, evolutivas e corretivas consomem grande parte do tempo. Como uma aplicação que foi construída fazendo uso de boas práticas, incluindo a construção de sub-rotinas da forma correta, pode reduzir o tempo das manutenções?

A criação de uma sub-rotina tem alguns propósitos. Entre eles, o principal é o reuso de um trecho de codificação e, consequentemente, tornar a manutenção pontual. Se um trecho de código é responsável por um conjunto de ações e outras partes apenas reutilizam essa sub-rotina, qualquer alteração tem impacto em toda aplicação, o que facilita a manutenção de qualquer tipo.

Nas linguagens orientadas a objetos, a nomenclatura correta para sub-rotinas é método (as ações dos objetos) e não funções, como era na linguagem C. A partir de agora, o conteúdo das aulas fará menção a esta estrutura somente como MÉTODO.

Um método em Java possui um nome, um tipo de retorno e argumentos. Assim como você declarava as funções no C, o nome e os argumentos de um método também são reconhecidos como **a assinatura de um método**. Neste momento do curso, você irá conhecer, inicialmente, somente os métodos **static**. Estes métodos que pertencem à classe são invocados ou acessados diretamente pelo nome da classe. Guarde esse conceito, você irá precisar dele em pouco tempo. A Figura 1 mostra em detalhes cada elemento do método **static**.

Figura 1 - Anatomia de um método static.



A seguir, na Tabela 1, observe um exemplo de como seria um programa que faz uso de um método para calcular a soma de dois valores em Java:

Tabela 1 – Exemplo de soma com método.

Programa em Java: Somador.java

```
public class Media{
    public static void main (String args[]){

        double valorUm = 10;
        double valorDois = 3;

        //invocando um método
        double somar = somar(valorUm, valorDois);
        System.out.println(somar);
    }

    //os metodos devem ficar dentro da classe e  fora do main

    public static double somar(double valorUm, double valorDois)
        return (valorUm + valorDois);
    }
}
```

A partir do exemplo, foi possível observar onde o método deve ser implementado (fora do main e dentro da classe), quanto a sua assinatura, e como deve ser chamado.

Um fator importante para o uso de métodos é a passagem de valores para os métodos. Você se lembra da história de passagem por cópia/valor e passagem por referência? Pois bem... Para tipos primitivos, o Java trabalha com passagem por cópia do valor, já para os objetos (um array, por exemplo) a passagem se dá por cópia, mas da referência. Confundiu-se, né? Java não possui aritmética de ponteiros, por isso é dito cópia da referência. O que

você precisa saber é o seguinte: se você passar de um tipo primitivo como parâmetro para um método, e esse método alterar o valor passado, o valor original não sofrerá alterações. Agora, se você passar um objeto e fizer alguma alteração neste objeto, ele sofrerá alteração. Agora, você pensou: "Isso é passagem por referência, pelo menos o comportamento é igual". Você está certo, o comportamento é o mesmo, mas conceitualmente não se pode falar por referência como é no C, pois não consigo realizar aritmética sobre a referência passada.

Para Refletir

Dentro do exposto, considere que, ao passar um *array* para um método e este método em uma determinada posição altera o valor. Ao terminar a execução do método, o *array* passado terá sido alterado?

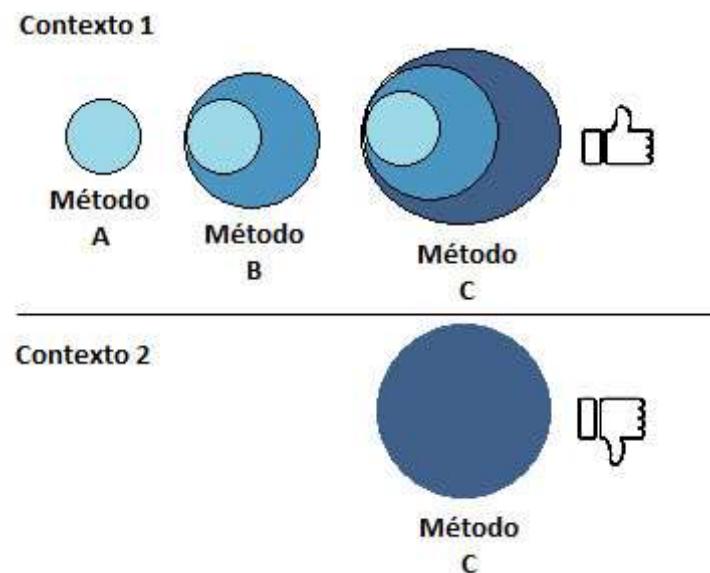
Ao passar um **objeto** como argumento, é passada uma **cópia da referência** do objeto. Consequentemente, qualquer **manipulação nos atributos** refletirá na referência original. Mais detalhes sobre essa diferença serão tratados nas próximas aulas.

Agora, dê uma pausa e pratique. **Faça os exercícios 1 e 2 do item “Na Prática”.**

Foco no Reuso

Uma das principais características da programação orientada a objetos é a capacidade de reutilização de componentes. Essa reutilização deve começar em nível de métodos. Para você começar a entender este conceito, você precisa aprender a criar métodos coesos e que sejam reaproveitáveis. Ao criar um método de alta complexidade, você deve criar vários métodos de baixa complexidade que, combinados por outro método, pode se tornar um método mais complexo. A regra é dividir para conquistar. A Figura 2, a seguir, apresenta esta ideia. Se o objetivo é construir um método complexo, construa pequenos métodos anteriormente para que também possam ser disponibilizados na classe.

Figura 2 – Evolução na construção de métodos.



O Desenvolvedor que programa pensando no **contexto 1** da Figura 2 fornece 3 métodos como serviço para uma classe. Se o desenvolvedor escolhe a estratégia do **contexto 2**, ele perde a oportunidade de deixar disponível serviços na classe que ele implementou os métodos.

Importante

Assista ao **vídeo 1** para ver na prática a implementação de métodos e algumas dicas importantes.

Vídeo 1

Prática Profissional - Programação orientada à objeto - Unid...



Importante

Assista ao **vídeo 2** para você observar na prática o exposto na Figura 2 no contexto prático de entrada de dados.

Vídeo 2

Prática Profissional - Programação orientada à objeto - Unid...



Você pode observar que foi mencionada **a sobrecarga** na explicação. A sobrecarga é a possibilidade de um método ter o mesmo nome que outro, mas com assinaturas diferentes. Os métodos têm a mesma finalidade, mas, dependendo do argumento, ele pode fazer uma ação diferenciada. No vídeo, três métodos chamados `lerDouble` foram criados, mas cada um fazia coisas diferentes, dependendo dos argumentos que eram passados.

Agora que você assistiu ao vídeo, e baseado no exposto até o momento, acesse o item “Na prática” e faça o exercício 3.

Java, esse tipo de representação é feito por meio de uma classe. "Mas espere aí, em uma classe pode colocar métodos e dados?" A resposta é: **PERFEITAMENTE**. À medida que você avançar no estudo do material, isso ficará mais claro.

A Tabela 2 é um exemplo de criação de uma entidade com a atribuição de valores e uso. Leia com atenção.

Tabela 2 – Listagem de criação de uma entidade e manipulação.

Programa em Java: ExemploEntidade.java

```
public class ExemploEntidade{
    public static void main (String args[]){
        // Na linha de baixo é criado seu 1º objeto
        /*L1*/ Aluno al = new Aluno( );
        /*L2*/ al.nome = "João da Silva";
        //o acesso aos atributos se dá por(.), assim como no C.
        al.matricula = "UC17000000";
        al.situacao = 'B';

        System.out.print("O nome do estudante: "+al.nome);

        //acalme-se isso é um operador ternário, entendeu?
        String texto = (al.situacao == 'B'?"BOLSISTA":"REGULAR");

        System.out.print("Situação do aluno: "+texto);
    }
}

//Definição de uma entidade chamada Aluno FORA da classe ExemploEntidade
class Aluno{
    String nome;
    int matricula;
    char situacao; //B - Bolsista R - Regular
}
```

Na codificação acima, em L1 foi criado um objeto **Aluno** para que você pudesse fazer uso. Na L2, por meio do nome da referência e (.), foi possível acessar os atributos daquele objeto. Percebe que o (.) serve para acessar os membros das classes, sejam métodos ou atributos? Guarde isso!

A Figura 3 faz uma representação do que acontece em memória quando se cria um objeto.

Figura 3 – Representação da memória ao criar um objeto exemplo.

Agora que você realizou os exercícios, no vídeo a seguir você encontra a correção. Mas veja bem, não adianta assistir à correção sem ter feito o exercício. Se você não a realizou ainda, volte lá e finalize. É bom para você! A correção serve para direcioná-lo à forma correta, não assista antes de esgotar suas tentativas de fazer, sozinho, os exercícios.

Importante

Assista ao **vídeo 3** para acompanhar a correção.

Vídeo 3

Prática Profissional - Programação orientada à objeto - Unid...

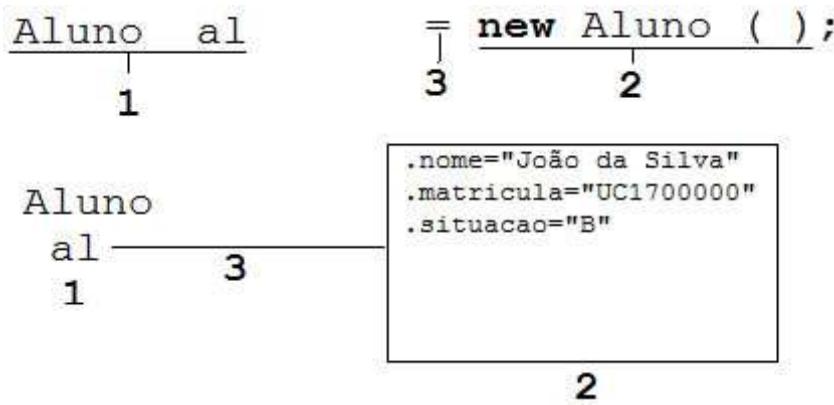


Conforme abordado no início da disciplina, é uma estratégia desse conteúdo aproveitar ao máximo o que você já conhece.

Os próximos dois tópicos fazem parte da transição do conhecimento. Se você já conhece orientação a objetos, sentirá falta de algumas coisas, entretanto, por estratégia didática, os conteúdos serão apresentados de forma gradativa e evolutiva. Siga em frente!

Criando Classes de Serviços

Como você pode observar na correção do exercício 3, a classe ficou muito extensa com vários métodos. Para que seja possível prover reuso dessas funcionalidades, temos que contextualizar os serviços. Que tal criar arquivos .java (Classes) que possamos colocar



Observe que:

- No passo 1, foi criada uma referência com 'al' do tipo Aluno (ainda não foi criado o objeto).
- No passo 2, foi criada uma área de memória do tipo aluno.
- No passo 3, a área de memória criada para o tipo aluno é atribuída à referência.

A partir deste ponto, por meio da referência 'al', é possível manipular a entidade aluno.

Para Refletir ⚡

O que aconteceria se em `ExemploEntidade.java` fosse alterado a L1 para: `Aluno al = null;`

A máquina virtual com a referência de um objeto declarado tenta acessar seus atributos, entretanto, ainda não existe uma alocação de memória para o objeto. Resumidamente, não foi acionado o operador `new` para alocação de memória para a referência declarada. Isto gera um erro no console chamado `NullPointerException`.

Array de Objetos

Dando continuidade aos estudos, já se perguntou como seria um *array* de objetos aluno? Se fosse necessário armazenar todas as informações de uma turma de alunos, seria necessário um *array* do tipo aluno.

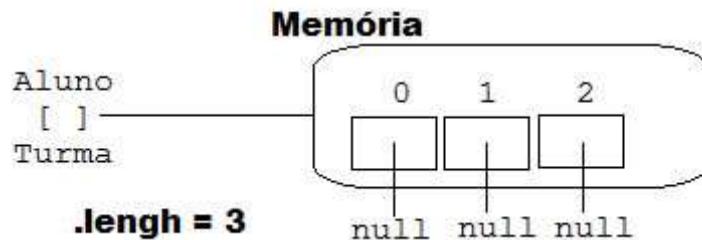
Ex: `Aluno [] turma = new Aluno [3];`

O exemplo de código acima é uma declaração de um tipo *array* capaz de armazenar referências do tipo aluno. Em cada posição desse *array* haverá uma referência do tipo Aluno. A Figura 4, a seguir, faz uma representação de memória para instanciação do exemplo.

Figura 4 – Representação da memória na instanciação de um *array*.

Código

```
Aluno [ ] turma = new Aluno[3];
```



Quando são instanciados, os objetos são inicializados com null. Lembra-se de que na aula passada os *arrays* de inteiros eram inicializados com zero? Pois bem, isso só vale para os *arrays* de tipos primitivos.

Quando se instancia um *array* de objetos, seu conteúdo fica igual ao representado na Figura 4. Logo, não é possível "setar" um atributo nos "alunos" deste *array*. Fazer `turma[0].nome = "José"` não é correto, pois, conforme apresentado na Figura 4 `turma[0]` retornará null. E ação de referência (.) em null é erro no Java, é lançada uma exceção.

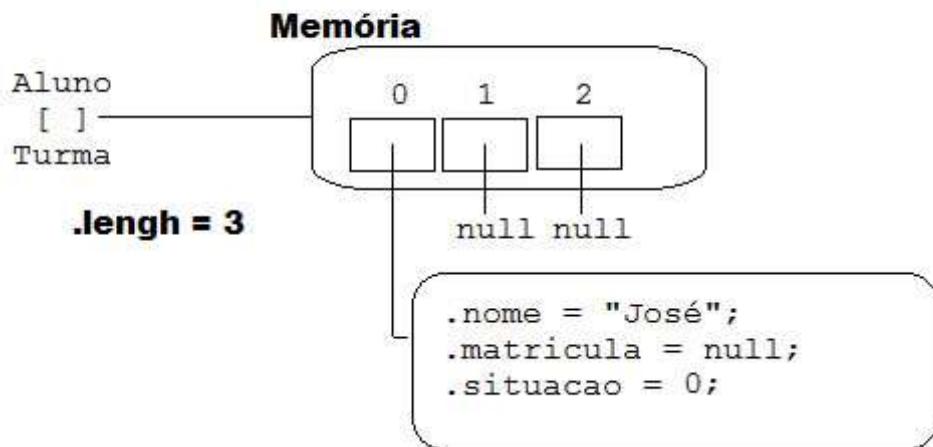
Mas o que fazer então? Neste caso, que *array* ainda não foi populado, você deve inicializar a posição antes de manipular.

Observe a Figura 5 a seguir, que representa a memória após a inicialização:

Figura 5 – Representação da memória após a inicialização de um objeto.

Código

```
Aluno [ ] turma = new Aluno[3];
turma[0] = new Aluno( );
turma[0].nome = "José";
```



Perceba que agora existe uma memória alocada para a posição 0 do *array*, pois foi acionado `turma[0] = new Aluno()`. Entenda, a partir de agora, toda vez que você encontrar o operador '**new**' você deve pensar: "Foi criada uma nova área de memória do tipo que está à frente do **new**".

Para Refletir

Sobre instanciar objetos, qual a diferença dos trechos de códigos 1 e 2 a seguir:

1 - `new Aluno();`

2 - `new Aluno[3];`

Atenção! Você não deve confundir a instanciação de um objeto (trecho de código 1) com a instanciação de um *array* de objetos (trecho de código 2).

Agora, um último detalhe sobre os *arrays*: Como é a assinatura de um método que receba um *array* do tipo *aluno* ou que retorne um *array* de *aluno*?

Exemplo de retorno: `public static []Aluno fazAlgo (....) { }`

Exemplo de passagem: `public static void fazAlgo (Aluno []turma) {}`

todos os serviços daquele contexto? Imagine uma classe Leitor.java com todos os métodos de leitura. Imagine uma classe Matematica.java com todos os métodos de operações matemáticas. Desta forma, as referidas classes seriam especialistas em fornecer serviços de acordo com o que elas possuem como contexto, entendeu?

Importante

Assista ao **vídeo 4** para você compreender bem esse processo.

Vídeo 4

Prática Profissional - Programação orientada à objeto - Unid...



Agora você percebeu que foram criadas classes que poderão ser usadas em outras situações. Isso facilita a manutenção e evolução de sistemas. O que você deve perceber é que as classes agora têm ações (métodos que fazem coisas) específicas à natureza delas.

Criando Classes de Dados

Na computação, às vezes, é preciso criar uma representação com informações de diferentes tipos de dados. Por exemplo, para representar uma entidade aluno de uma turma, em uma determinada abstração (para um propósito), poderíamos precisar de nome, matrícula, cpf e idade. Perceba que há uma heterogeneidade de dados para compor a entidade aluno. Para resolver isso, precisamos definir uma estrutura de dados heterogênea, uma entidade complexa. Na linguagem C, esse recurso é conhecido como ***struct***. No

Gostou do vídeo? Agora, pratique, faça você mesmo! A repetição vai te levar à destreza. Siga em frente!

Finalizando... C

Nesta aula, você começou sua transição para orientação a objetos. Você foi apresentado aos métodos e entendeu que este recurso é fundamental no fornecimento de serviços por parte das classes. Compreendeu a importância de deixar as classes separadas com seus serviços.

Você também observou que é possível criar classes para representar dados, informação e visualizou a instanciação de objetos.

Nos vídeos desta aula, procurou-se apresentar boas práticas de programação e, principalmente, algumas técnicas para você se tornar um programador de objetos com qualidade, criando serviços com qualidade.

Agora, vá até o item “Na prática” e faça todos os exercícios da aula. Entenda... para você conseguir se tornar um programador orientado a objetos, deve primeiramente ser um bom programador de métodos. Na próxima aula, começará, de fato, a sua vida na orientação a objetos, pois estudará dois de seus pilares importantes. Espera-se que sua preparação tenha sido excelente até aqui. Até a próxima!

Importante

Assista ao **vídeo 5**, para observar uma implementação prática do que foi apresentado sobre os *arrays de objetos*.

Vídeo 5

Prática Profissional - Programação orientada à objeto - Unid...



Na Prática

"Prezado(a) estudante,

Esta seção é composta por atividades que objetivam consolidar a sua aprendizagem quanto aos conteúdos estudados e discutidos. **Caso alguma dessas atividades seja avaliativa, seu (sua) professor (a) indicará no Plano de Ensino e lhe orientará quanto aos critérios e formas de apresentação e de envio.**"

Bom Trabalho!

Atividade 01

^

Implemente um programa em Java que calcule o juro de uma dívida que você contraiu no mês passado no crediário de uma loja. A taxa de juros mensal e o valor da dívida serão fornecidos pelo usuário. A dívida deve ser calculada por método criado por você chamado: **calcularDivida** e deve receber como parâmetro os valores necessários para o cálculo e retornar a referida dívida.

Atividade 02

^

Implemente um programa em Java que receba a temperatura em graus Celsius e apresente-a convertida em graus Fahrenheit. A fórmula de conversão é: $FAR = (9 * CEL + 160) / 5$, sendo FAR a temperatura em Fahrenheit e CEL em Celsius. Esta conversão deve ser feita por meio de um método com a seguinte assinatura: **public static double fahrenheit(double celsius)**. O método deve receber a temperatura em Celsius e retornar em Fahrenheit.

Atividade 03



Você deve implementar uma calculadora. O usuário informará um valor, um operador e outro valor. Execute a operação e apresente o resultado. Faça uso do maior número de métodos que conseguir visualizar (na entrada de dados, nas operações). As operações disponíveis para esta calculadora são: adicionar, subtrair, dividir, multiplicar e expoente. Para o cálculo do expoente, pesquise no [hiperlink](#) .

Atividade 04



Implemente um programa em Java que calcule o somatório de um número inteiro positivo recebido pelo usuário (efetue a validação para garantir esta premissa). Um método chamado **somatório** deve ser criado por você. Como parâmetro, ele deve receber um número, e o retorno deve ser a soma de todos os números anteriores até o zero. Por exemplo: se for passado 3 para o método, deve retornar a soma de $0 + 1 + 2 + 3$.

Atividade 05



Você deve fazer a implementação de uma classe chamada MyMath.java, que deverá ter quatro métodos conforme a listagem a seguir:

- Receber um número inteiro como argumento e calcular o fatorial.
- Receber dois valores como argumento e retornar o número que for maior.
- Receber um valor inteiro como argumento e retornar se ele é um número par.
(retorne *true* ou *false*)
- Receber um número inteiro e retornar se ele é um número primo matemático (desafio).

Faça um programa principal para testar sua classe implementada com valores atribuídos a sua escolha.

Atividade 06



Escreva um programa Java que lê um valor n inteiro e positivo e que calcule a seguinte soma por meio de um método chamado somar S: $S := 1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$

Atividade 07



Faça um programa que receba 5 mil dados do usuário do tipo inteiro.

Sabe-se que valores negativos não são aceitos. Após receber esses valores e popular o array, imprima na saída padrão: a média dos valores, quantos valores são ímpares e todos os valores que foram informados. Para esta solução, utilize o maior número de métodos visualizados.

Atividade 08



Crie uma entidade **aluno**, com nome e duas notas. Receba do usuário os dados deste aluno e, na sequência, apresente todos os dados deste aluno. Apresente também a média dele e se está aprovado ou reprovado. Crie o maior número de métodos que conseguir visualizar.

Atividade 09



Uma escola deseja fazer um cadastro dos seus alunos. Um aluno possui nome, matrícula e situação, que pode ser regular ou bolsista. O usuário informará quantos alunos existem em uma turma. Você receberá os dados de todos os alunos e, na sequência, você deve apresentar todos os alunos informados, a quantidade de alunos regulares que existe e a quantidade de alunos bolsistas.

Atividade 10



Um piloto tem nome, escuderia e tempo de volta (em minutos apenas não pode ser menor que 1 nem maior que 60). Faça um programa Java que receba os dados dos pilotos de uma corrida. Ao final mostre o seguinte:

- O piloto mais rápido.
- O piloto mais lento.
- A média dos tempos.
- **DESAFIO:** todos os pilotos ordenados pelo tempo. (Use o bubblesort)

Assista ao vídeo "[Bubble-sort with Hungarian](#) ", que explica o método de ordenação de forma não convencional.

Referências

- BOOCH, G. **Object-Oriented Design with Applications**, Benjamin-Cummings, 1991.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – guia do usuário**. 2. ed. Rio de Janeiro: Campus. 2006.
- CARDELLI, L.; WEGNER, P. **On Understanding Types, Data Abstraction, and Polymorphism**. ACM Computing Surveys (CSUR). vol. 17, pp. 471-523. 1985.
- DEITEL H. M.; DEITEL, P. J. **Java, como programar**. 6. ed. Porto Alegre: Bookman. 2006.
- DICIONÁRIO AURELIO. 2017. Disponível em:
<https://contas.tcu.gov.br/dicionario/home.asp> Acesso em: 19 mar. 2017.
- ERICH, G. et al. **Padrões de projeto**: soluções reutilizáveis de software rientado a objetos. Porto Alegre: Bookman. 2000.
- HORSTMANN, C. S.; CORNELL, G. **Corejava 2 – Volume I – Fundamentals**. São Paulo: Makron Books. 2010.
- NEWRELIC. **The Most Popular Programming Languages of 2016**. 2016. Disponível em:
<https://blog.newrelic.com/2016/08/18/popular-programming-languages-2016-go>. Acesso em: 9 Mar 2017.
- NIEMEYER, P.; KNUDSEN, J. **Aprendendo Java**. Rio de Janeiro: Campus. 2000.
- ORACLE. **Java Licensing Logo**. 2017. Disponível em:
<http://www.oracle.com/us/technologies/java/java-licensing-logo-guidelines-1908204.pdf>. Acesso em: 13 mar. 2017.
- SIERRA, K.; BATES, B. **Certificação Sun para Programador JAVA 5 Guia de Estudo**. Rio de Janeiro: Alta Books, 2006.
- SILBERSCHATZ, A; GALVIN, P. B.; GAGNE, G. **Sistemas operacionais com Java**. 6. ed. Rio de Janeiro. Editora Campus, 2004.

- STUCKEY, P. J.; SULZMANN, M. **A theory of overloading**. International Conference on Functional Programming. Proceedings of the seventh ACM SIGPLAN international conference on Functional programming. Pittsburgh, PA, USA, 2002. pp. 167-178.
- WIKIPÉDIA. Desenvolvido pela Wikimedia Foundation. Conteúdo sobre o **Ambiente de Desenvolvimento Integrado**. 2017. Disponível em: <https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado>. Acesso em: 5 mar. 2017.