

Juntando as Coisas: Criando Objetos

Apresentação

Até o momento, foi objetivo do conteúdo da disciplina familiarizar você com a sintaxe do Java e, em paralelo, prepará-lo para compreender os conceitos da orientação a objetos na prática. Esta base serve para que, de uma forma sutil, esses conceitos possam fazer parte da sua forma de pensar ao resolver problemas computacionais. Imagine você sem intimidade com o Java tentando aplicar na prática um conceito novo? Você teria raiva da linguagem e aprender orientação a objetos poderia ser apenas uma tentativa.



Fonte: <https://goo.gl/EAkHXy> ↗

Nesta aula, você mergulhará, de fato, na orientação a objetos. Conhecerá a diferença entre uma classe e um objeto. Conceitos importantes como agregação e composição. Aprofundará seus estudos na construção dos objetos. Entenderá o relacionamento TER dos objetos, a troca de mensagens entre objetos e a responsabilidade dos objetos.

Finalmente, conhecerá os dois primeiros pilares da orientação a objetos: abstração e o encapsulamento, e o melhor, tanto na teoria quanto na prática.

Bons estudos!

Desafio



Fonte: <https://goo.gl/3hQUKB> ↗

Com a chegada das eleições, é preciso gerenciar candidatos que fazem parte deste grande circo. Deverá ser feito um sistema em Java usando os recursos vistos até o momento na disciplina.

O candidato possui nome, número de filiação, se a candidatura é para deputado federal, distrital ou senador. Cada candidato tem direito a uma verba de campanha, de acordo com o seu tipo de candidatura. Se o candidato almejar candidatura para deputado distrital, a verba destinada a ele é de R\$ 170.000,00, caso almeje deputado federal R\$ 250.000,00, e, por fim, caso almeje senador R\$ 350.000,00. Outra característica de candidato é se ele já foi ou não reeleito, caso ele já tenha sido reeleito a verba de campanha será dobrada.

Sabe-se que será informado um número fixo de candidatos, pergunte ao usuário a quantidade e receba os dados do mesmo. Após o recebimento dos dados, apresente, sem a necessidade de um menu de opções, as seguintes informações:

- a. Apresente todos os candidatos informados pelo usuário.
- b. Apresente todos os candidatos que já foram reeleitos.
- c. Apresente o total gasto com os deputados que não foram reeleitos.
- d. Apresente o total gasto com todos os candidatos.
- e. Apresente todos os candidatos que se chamam Tiririca.
- f. Apresente também a média de custo com os candidatos.

Conteúdo

Classes e Objetos

Desde criança, você enxerga o mundo por intermédio de objetos. "Olha o totó, olha o avião, olha o carro...". Você via o mundo cheio de objetos e que faziam alguma coisa. Apertava um brinquedo e ele emitia um som, o cachorro (totó) que latia, o avião que voava, etc.

À medida que crescia, seus pais começaram a te perguntar: "Qual a cor daquele carro?" Você trazia muita felicidade ao acertar as cores. Até que você entendeu que tinham vários objetos do mesmo tipo, mas com "valores" diferentes. Um carro branco da marca BMW, um carro amarelo da marca GM, e muitos outros!

Criar uma classe **Carro** é criar uma abstração de uma entidade, ou seja, é declarar características (cor, marca) e ações (acelerar, ligar e desligar). Instanciar uma classe **Carro** é criar objetos (um carro amarelo, BMW que acelera, liga e desliga).

Na aula passada, foi criada uma classe **Aluno** (uma abstração de aluno para resolver um problema computacional) com alguns atributos. Ao **instanciar** um aluno (`Aluno a = new Aluno();`) é criado um **objeto** de fato.

Quando um objeto é criado (instanciado), ele terá capacidade de ter ou de fazer coisas a partir do que foi declarado em uma classe. Por exemplo, se você declarou que uma classe **aluno** possui um nome, consequentemente, será capaz de atribuir um nome para o objeto **aluno**.

O Paradigma Orientado a Objetos (POO) é pensar e organizar sua solução assim como no mundo real. Isso, do ponto de vista teórico, faz com que seja mais fácil a manutenção, segundo BOOCH (1991). Ele define objeto como o seguinte: "Um objeto tem um **estado, comportamento e identidade**; a estrutura e o comportamento de objetos semelhantes são definidos em sua classe comum; os termos instância e objetos são intercambiáveis" BOOCH (1991), grifo nosso.

O estado são valores atribuídos aos atributos. Por exemplo: um aluno com uma nota 10.

Já, o comportamento é uma ação definida. O cálculo da média dentro da classe aluno.

Por fim, a identidade é abstração da classe Aluno. Aluno é a identidade.

Na programação estruturada, tem-se uma visão estritamente modular. Os programadores criam funções e parametrizam os dados para que as operações aconteçam. Dados de um lado e funções de outro.

Já, na programação orientada a objetos, dados (atributos) e funções (métodos) repousam no mesmo lugar. Uma classe é declarada com atributos e métodos.

Imagine uma abstração de aluno com: nome e duas notas. Uma estratégia estruturada para resolução deste problema, por exemplo, seria:

```
double media = calcularMedia(aluno.notaUm, aluno.notaDois);
```

Perceba que o objeto se compõe e a responsabilidade é passada para a função de fazer o cálculo.

Por outro lado, na visão orientada a objetos, uma estratégia para realizar a média de um aluno seria diferente. Conforme abordado, dados e operações (que atuem sobre os dados) repousam no mesmo lugar. A média pertence a que abstração? DO ALUNO. O aluno na verdade tem nome, duas notas e sua média. Na orientação a objetos, as regras de negócio (operações/métodos) devem estar onde os dados estão.

A média de um aluno não é um atributo, mas sim um método, pois o resultado da média depende de dois atributos que estão em aluno. Mas entenda, a média é do aluno, então deve estar em aluno. Veja o exemplo de chamada por meio de uma referência de aluno (al) anteriormente instanciada:

```
double media = al.getMedia();
```

O cálculo será efetuado a partir das notas presentes no aluno. Compreendeu?

Uma turma tem código, professor e vários alunos.

Perceba que nesse sentido, o verbo TER significa ser um atributo ou método dentro da classe.

Imagine que se deseja fazer um sistema que controle as turmas de uma escola e que, para os clientes, o importante é o seguinte: Sabe-se que uma turma tem código, professor, vários alunos, e se é ou não de laboratório. Um professor, por sua vez, tem nome, salário e matrícula. Já, dos alunos é preciso saber nome, matrícula e duas notas. Como ficaria a abstração destas classes? Observe com atenção o exemplo de código da Tabela 1:

Tabela 1 – Abstração de uma classe Turma em Java.

Programa em Java:Programa.java

```
public class Programa{  
    public static void main(String args[]){  
        Turma t = new Turma();  
        t.codigo = "MIB102";  
        t.professor = new Professor();  
        t.professor.nome = "José da Silva";  
        t.professor.matricula = "123456";  
        .  
        .  
        //fim  
    }  
}  
class Turma{  
    String codigo;  
    Professor professor;  
    boolean isLaboratorio;  
    Aluno []alunos;  
}  
class Professor{  
    String nome;  
    String matricula;  
    double salario;  
}  
class Aluno{  
    String nome;  
    String matricula;  
    double notaUm;  
    double notaDois;  
}
```

Importante

- Resumidamente, para quem está começando a desenvolver no paradigma orientado a objetos, é importante compreender o seguinte: os métodos que dependem dos dados de um objeto devem estar onde os dados estão. Assim, no exemplo anterior, **Média** deve estar dentro da classe aluno, pois ele depende e atua sobre as notas do aluno.
- Assista ao **vídeo 1** para você acompanhar a diferença de organização do pensamento no paradigma orientado a objetos em relação ao estruturado.

Vídeo 1

Prática Profissional - Programação orientada à obje...



Relacionamento TER

Dois verbos regem a organização do pensamento no paradigma orientado a objetos. O verbo TER e o SER. O verbo SER será tratado na Unidade 2 por estar relacionado a um importante pilar da orientação a objetos, que é a Herança.

Em relação ao verbo TER, **um objeto pode ter outros objetos**. Por exemplo: um objeto carro tem vários acessórios. Um objeto nota fiscal tem vários itens de nota. Um time de futebol tem um nome, uma data de fundação, uma comissão técnica e vários atletas.

Para Refletir ⚡

Diante do exposto nos tópicos anteriores, sabendo que um sistema deseja armazenar todos os projetos de pesquisa que acontecem em uma instituição. Em que Entidade (Classe) na sua abstração estaria um método que calculasse o gasto com cada Projeto? E em que Entidade estaria um método que calculasse todo o custo com os projetos da Instituição?

Para colocar as ações nos objetos corretos, basta você colocar as ações onde os dados estão. Logo, o custo de um projeto deve estar na classe projeto. Já, o custo de todos os projetos de uma instituição deve estar na classe Instituição.

A partir do que foi abordado, cabe destacar que, sempre que você partir para implementação de uma classe, você deve se fazer perguntas como as que seguem: onde seria implementado o método de cálculo de contribuição sindical de um professor (sabendo que esse imposto é 1% do salário)? R: Na classe Professor, pois é lá que está o salário e a contribuição sindical, que é do professor.

Onde seria implementado o método de cálculo de média de um Aluno? R: Em Aluno, pois a média é do aluno e é na classe aluno que estão as notas.

Onde seria implementado o método de cálculo de média da Turma? R: Turma, pois a turma tem todos os alunos.

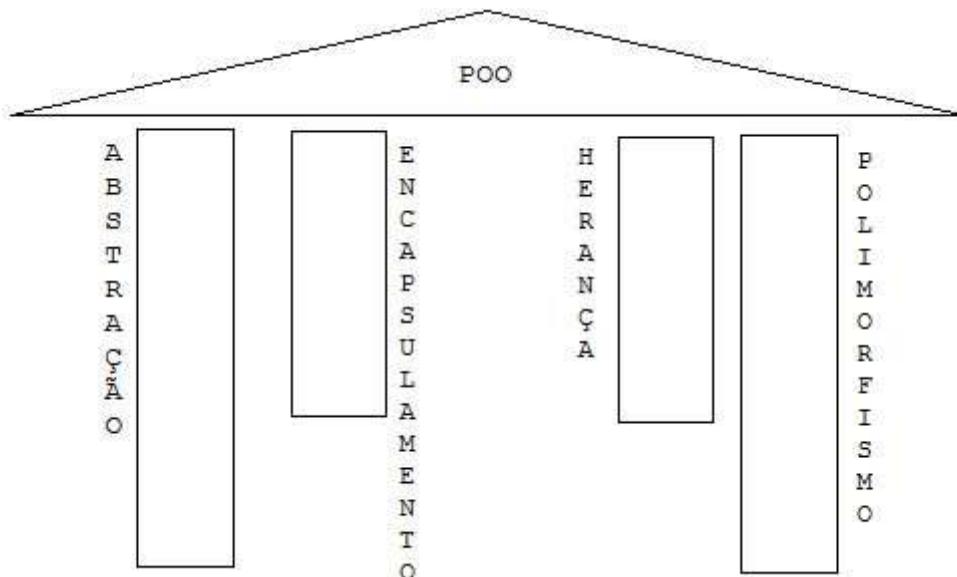
Em linhas gerais, comece definindo o que é atributo (dado) e o que é método (calculado) na sequência descubra onde cada coisa deve estar. É um excelente começo.

Pilares da Orientação a Objetos

Para "organizar" o pensamento voltado à programação orientada a objetos, você tem que levar em consideração os quatro pilares da orientação a objetos: **Abstração, Encapsulamento, Herança e Polimorfismo**.

Assim como os pilares de uma casa, que dão sustentação ao telhado, os pilares da orientação a objetos citados dão sustentação ao paradigma. Se um dos pilares falhar, o telhado vem ao chão. Não adianta fazer uma abstração perfeita se o encapsulamento for ruim e vice-versa. Nenhum pilar pode falhar, lembre-se disso. A Figura 1 representa os pilares da orientação a objetos.

Figura 1 – Representação dos pilares da orientação a objetos.



Abstração

A **Abstração** é você retirar de um cenário estritamente o necessário, tão somente o necessário, para dar **identidade** (definir o nome da classe), **características** (atributos) e **comportamentos/ações** (métodos) aos objetos.

Uma entidade carro, por exemplo, tem uma abstração (características e ações) completamente diferente para um agente do departamento de trânsito do que teria para um colecionador de carros esportivos. Concorda?

Esse pilar existe para que a abstração das entidades seja focada no problema a ser resolvido. Imagine um programador definindo uma classe pessoa baseado no mundo real? Passaria o resto da vida criando atributos.

Portanto, a sua abstração deve ser focada no problema a ser resolvido e garantindo a conformidade com os pilares da orientação a objetos.

Encapsulamento

O Encapsulamento é um pilar referente a **esconder detalhes**, isso é a chave. Dá para perceber isso na origem da palavra, não é verdade? Este pilar é levado em consideração tanto em relação aos **dados** quanto às **ações**.

Em relação às **ações**, você já observou a importância de colocar as ações no lugar certo e fazer com que os detalhes de implementação de um método fiquem encapsulados. É intuitivo e lógico para o programador chamar apenas:

```
double media = al.getMedia()
```

ou

```
double num = professor.getImpostoRenda().
```

Você não deve fornecer detalhes de implementação das operações disponíveis em seus objetos. O cálculo do imposto de renda é complexo, mas pouco importa, queremos apenas chamá-lo e ele deve retornar o valor que se propõe a calcular; por isso, seus métodos devem fazer somente o que se propõem a fazer e estar bem definido. Imagine se você instanciar um objeto e, ao acionar um método, ele fizer algo que você não espere, mesmo que seja a mais? Você terá que conhecer bem os detalhes da implementação... Quando isso acontece, significa dizer que este método está com baixa coesão e alto acoplamento e, consequentemente, quebrou o pilar do Encapsulamento. Quer um exemplo?

Muitos programadores iniciantes quando desafiados a criarem uma solução que calcule e apresente a média de duas notas costumam acoplar este tipo de solução.

Como o código de exemplo abaixo:

```
public void calcularMedia(double nota1, double nota2){  
    System.out.println((nota1+nota2)/2);  
}
```

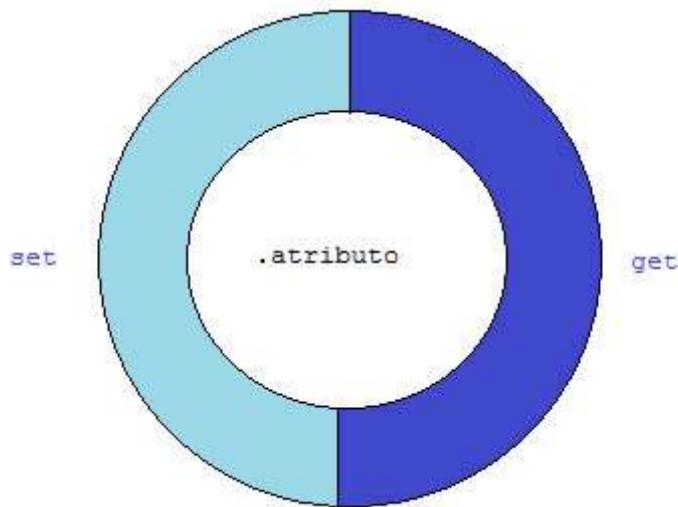
Perceba que este método não está coeso, pois faz atividades a mais do que se propõe. Ele deveria apenas realizar a média e retornar o valor, e não diretamente realizar a impressão do resultado. A solução está acoplada ao uso de *console*. Se fosse preciso apresentar esta média em uma tela mais trabalhada, usando uma API Swing do Java, não seria possível utilizar este método.

Só para lembrar, o cálculo de Imposto de Renda faz uma série de cálculos em um amplo cenário. Esse é um bom exemplo de que um método apenas não pode ser usado para criar a solução. Lembre-se disso na aula passada. Devem ser criados vários métodos para chegar ao cálculo final. Por exemplo, ter um método que calcule somente gastos com saúde. Geralmente, os clientes de sistema pedem relatórios de todos os tipos e, quase sempre, se seu programa estiver bem programado, a resposta a sua área de negócio, que é quem paga seu salário, será satisfatória e entregará muito valor.

Já, sobre os **dados**, de acordo com o pilar do encapsulamento, não podemos permitir acesso direto aos dados. É preciso esconder os dados dos outros objetos.

Um atributo é acessado com dois propósitos: atribuir um valor ou obter um valor para fazer algo com ele. Essas duas operações devem ser feitas por intermédio de métodos. Conhecidos como **métodos acessores**. Que por padrão devem se chamar get(nome do atributo) e set(nome do atributo). A Figura 2 representa o papel dos métodos acessores em relação aos atributos.

Figura 2 – Papel dos métodos acessores.



Para cada atributo, existirão dois métodos na classe para fazer as duas operações de colocar um valor (set) e obter um valor (get).

Além disso, outra ação deve ser feita. A visibilidade do atributo deve ser modificada para impedir que objetos externos acessem o atributo. Todos os atributos da classe devem ter seus acessos modificados para private.

Logo, se na classe **Aluno** que possui um nome colocarmos: `private String nome;` Fora da classe que está o atributo não será mais possível fazer a seguinte codificação: `al.nome = "João"` (erro de compilação), pois o atributo nome não estará mais visível. Para colocar um nome no aluno seria necessário o seguinte: `al.setNome("João")`. Para obter: `String nome = al.getNome();`

Dessa forma, somente os métodos do objeto ficarão visíveis. O seu objeto estará encapsulado com todos os detalhes escondidos.

System por exemplo. Percebeu o asterisco, né? Significa que ele tem inúmeras classes dentro deste pacote e estamos fazendo menção a todas. Este pacote possui classes importantes como: `String` , `Math` , `Object` , etc.

Para você praticar e entender a importância dos pilares, dois objetos importantes no Java serão utilizados, para você se aproveitar dos seus recursos e, principalmente, perceber a importância de se programar corretamente nos pilares citados. Esses objetos são: `java.lang.String` e `java.util.ArrayList` .

Concentre-se em não só saber o que cada método faz, pois para isso tem a documentação dos objetos que será apresentada a você depois. Tente observar a forma como foram programados os objetos, perceba que oferecem serviços com coesão e oferecem vários serviços por meio de sobrecarga. Tente imitar o Java!

Objeto `String`

O objeto `String` fornece uma API (métodos disponíveis para que outros programadores utilizem) que possibilita diversas operações sobre as cadeias de caracteres, como: ter uma `String` toda em caixa alta ou baixa, comparar se um texto é igual ao outro independente de caixa, verificar se um texto começa com alguma palavra, verificar qual é o índice de um determinado caractere, entre outros.

Além de conhecer esses métodos, você tem que entender que o programador que construiu esta classe se preocupou com uma coisa: FORNECER SERVIÇOS. E esses serviços serão utilizados por toda a comunidade desenvolvedora Java. Já imaginou todos os programadores tendo que implementar estas funcionalidades toda vez que fossem trabalhar com texto? Por outro lado, se esses serviços implementados fossem mal feitos e sem coesão, a comunidade conseguiria usar? NÃO. A Tabela 2 traz a lista dos principais métodos da entidade `String` , uma breve explicação do que faz e um exemplo de uso.

Tabela 2 – Principais métodos da API String.

Método	Descrição
<code>public char charAt(int index);</code>	Retorna o char que se encontra na posição que tiver sido passado como argumento. O índice começa de 0.

Importante

Assista ao **vídeo 2** para você conhecer mais detalhes sobre o encapsulamento.

Vídeo 2

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



Interagindo com Objetos do Java

Quando uma linguagem é criada, algumas operações consideradas essenciais à linguagem vêm nativamente. Por exemplo, o `scanf` é uma função disponibilizada pela linguagem C que é nativa para fazer leitura de dados e está em uma biblioteca chamada `#include <stdio.h>`. Você já conheceu um pacote do Java que lhe permitiu fazer entrada de dados que é a `java.util.Scanner`.

Toda vez que você precisou fazer uma leitura pela console, você teve que importá-lo. Existe um pacote que implicitamente já é importado por todas as classes que é o `java.lang.*`. Por isso que você nunca teve que importá-lo para usar a classe `String` ou

Método	Descrição
Exemplo:	

```
String linguagem = "Java";
char c = linguagem.charAt(2); // será retornado 'v'
```

Método	Descrição
boolean endsWith(String suffix)	Testa se uma String termina com um determinado sufixo que foi passado como argumento.
Exemplo:	

```
String texto ="Linguagem de Programação Java";
boolean b = texto.endsWith("Java");//será retornado TRUE
```

Método	Descrição
boolean equals(Object anObject)	Retorna true se a String que for passada for exatamente igual. Sim, é um objeto como parâmetro se for passado outro tipo, como um Aluno. Retorna falso.
Exemplo:	

```
String nome = "Java";
boolean b = nome.equals("java"); // será falso, pois uma tem J maiúsculo
b = nome.equals("Java"); // será verdadeiro, pois são exatamente iguais.
```

Método	Descrição
int length()	Retorna o tamanho da String.

Método	Descrição
Exemplo:	

```
String nome = "Java";
int tam = nome.length(); // Retornará o tamanho atual da String: 4.
```

Método	Descrição
boolean startsWith(String prefix)	Testa se uma String começa com um determinado prefixo que foi passado como argumento.
Exemplo:	

```
String nome = "Java";
boolean b = nome.startsWith("Ja");// retorna TRUE.
```

Método	Descrição
String toUpperCase()	Retorna uma NOVA String com toda a caixa alta.
Exemplo:	

```
String nome = "Java";
String upper = nome.toUpperCase(); // Retorna JAVA.
```

Método	Descrição
String toLowerCase()	Retorna uma NOVA String com toda a caixa baixa.

Método	Descrição
Exemplo:	<pre>String nome = "JaVa"; String lower = nome.toLowerCase(); // Retorna java.</pre>

Sobre os objetos **String**, cabe destacar dois aspectos importantes. Primeiramente, as Strings no Java são imutáveis, ou seja, você não modifica uma String, você sempre cria uma nova String. Ao chamar **toUppercase**, ele retorna uma nova String e não modifica a original. Ao concatenar uma String com outra String, é criada uma nova String.

Por fim, nunca use **==** para comparar se uma **String** é igual a outra, pois este operador aplicado aos objetos não analisa o conteúdo do objeto, mas somente o endereço de memória que ele está ocupando naquele momento. Logo, você pode comparar uma String exatamente igual a outra que retornará falso.

variável.

Fazendo uma análise dos *arrays*, somente a limitação de possuir um tamanho fixo incomoda. Pois é muito conveniente acessar uma posição específica e ter a segurança de armazenar os mesmos tipos de dados em uma estrutura.

O *ArrayList* faz isso para você. Ele possui tamanho variável, ou seja, enquanto existir espaço em memória, ele crescerá. Além disso, permite o acesso indexado a uma determinada posição e permite ser homogêneo.

O programador que construiu o objeto *ArrayList* criou um método chamado *add* que ao passar um determinado objeto como argumento ele simplesmente adiciona no final da coleção o objeto passado. Perceba que é um método coeso e que **ocultou os detalhes de implementação** que, de certa forma, é complexo.

Na Tabela 3 é apresentado um exemplo de um programa que armazena Strings em uma coleção.

Tabela 3 – Exemplo de um programa com ArrayList.

Programa em Java:Programa.java

```
import java.util.ArrayList;//Import para usar o objeto ArrayList
public class Programa{
    public static void main(String args[]){
/*L1*/ ArrayList<String> times = new ArrayList();

        // adicionando elementos à vontade
        times.add("Botafogo");
        times.add("Flamengo");
        times.add("Fluminense");
        times.add("Vasco");

        //acesso indexado
        String time = times.get(2);// inicia de ZERO
        System.out.println(time+"\n"); //será apresentado: Fluminense

        //Para apresentar todos
/*L2*/ for(String elemento : times){
            System.out.println(elemento);
        }

    }
}
```

Importante

Assista ao **vídeo 3** para você conhecer mais detalhes sobre as Strings.

Vídeo 3

Prática Profissional - Program...



Visite também a [API da classe String](#) para conhecer outros recursos.

Agora, realize os exercícios 1 e 4 do item “Na prática”.

Objeto *ArrayList*

O mundo real é composto de objetos que interagem entre si e por meio dessas interações produzem algum tipo de resultado. Além disso, neste mesmo cenário, é comum encontrar coleções de objetos, que são objetos de mesmo tipo com valores diferentes. Uma coleção de carros, por exemplo.

Até o momento, você conheceu uma estrutura de dados chamada *array*, que possui as seguintes características: possui tamanho fixo, armazena informações do mesmo tipo (homogênea) e possui acesso indexado.

No mundo real, um conjunto de dados pode crescer. Imagine fazer uso de uma estrutura que limita a quantidade de carros que posso ter? Mesmo que seja definido um número grande, não é uma boa solução, concorda? O ideal é ter uma estrutura de tamanho

A forma de iteração da coleção é muito simples, não é verdade? Para cada vez que a estrutura de repetição for der um loop um novo elemento, é atribuído à variável temporária elemento. Esta forma de iterar uma coleção é conhecida como `forEach` .

Para Refletir

Você viu um jeito novo de imprimir os dados de uma coleção. Sabendo que o objeto `ArrayList` possui um método chamado `size()` que retorna o tamanho da coleção, com o auxílio do acesso indexado por meio do método `get (int index)` , tem como implementar uma outra forma de percorrer a coleção?

Da mesma forma como acontecia para percorrer um `array` com o indexador ' `i`' , você pode fazer na estrutura `ArrayList` para iterar.

Métodos Construtores

Na aula anterior, a Figura 2 fez uma abstração da memória quando se cria um objeto. O operador `new` , quando é acionado, invoca o método construtor do objeto em questão. Você pode se perguntar: "Mas que método é esse que eu nunca vi?". Pois bem, todos os objetos do Java possuem um método construtor padrão. Padrão significa sem argumentos.

Todo método construtor terá sempre o mesmo nome da classe. SEMPRE? Sim, sempre!

O construtor serve para executar algo que deseje no momento da criação do objeto e/ou solicitar dependências. A classe Scanner que você já usa é um exemplo disso.

Esta classe só é criada se passarmos o objeto que ela precisa para saber se a leitura acontecerá de um teclado ou de um arquivo, por exemplo.

Observe neste trecho:

```
Scanner scan = new Scanner(System.in)
```

O operador `new` está invocando o construtor do Scanner e passando para ele um argumento (`System.in`) que indica a leitura pelo teclado. Os engenheiros da Sun que criaram o Scanner sobrescreveram o método construtor para que, obrigatoriamente, na criação do objeto, fosse passado um argumento que é necessário para o funcionamento do objeto.

E você, como programador, poderia sobrescrever o construtor de Turma, para que só seja possível criar um objeto Turma se for passado, pelo menos, o Professor daquela turma. Afinal, existe turma sem professor?

Finalizando... C

Esta aula foi bem interessante, sim? Consegue perceber a evolução do conteúdo? Não será deixado nenhum conceito importante para trás, apenas será feito no momento certo.

*Você iniciou o seu contato com a orientação a objetos. A qualidade do seu código vem mudando e a forma de pensar também. Esse paradigma tem o foco na reutilização, mas para isso não adianta só implementar uma funcionalidade legal, é necessário implementar bem. Muitos exercícios são necessários. Nesta disciplina, o **como** fazer é importante, e não simplesmente imprimir uma saída correta. Ou seja, o caminho a ser percorrido para se chegar a um determinado resultado merece grande atenção.*

A Abstração e o Encapsulamento representam só o início desta jornada, pois ainda faltam a Herança e o Polimorfismo, que serão abordados na próxima unidade.

Nesta aula, você conheceu estruturas novas que potencializam seu desenvolvimento, como o ArrayList. Pode verificar também que a String tem diversos recursos a serem explorados.

Agora, é hora de praticar! Faça todos os exercícios que foram deixados para trás, poste suas dúvidas, assista aos vídeos. É fundamental que você se esforce para não deixar pendências para a próxima Unidade. Assim, poderá avançar nos estudos com segurança, sem dificuldades para aprender e aplicar os novos conceitos que serão apresentados.

Para encerrar esta Unidade com propriedade, assista ao [vídeo com a resolução de um exercício](#) ↗, que engloba todos os conceitos que estudou até o momento.

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

Crie um programa que seja uma calculadora, a partir de todos os conceitos que você aprendeu. Sua solução deve ter uma classe chamada **Calculadora**, que terá três atributos (dois operandos e um operador) e quatro métodos (que fazem as operações básicas: soma, subtração, multiplicação e divisão), que retornarão os resultados. Crie as leituras do usuário e faça a apresentação devida.

Atividade 02

Manipular *Strings* é uma atividade corriqueira de um programador. Crie um programa que fará a leitura de uma *String* do usuário e você deverá apresentar os seguintes resultados:

- a. A *string* informada em caixa alta.
- b. A *string* informada em caixa baixa.
- c. A quantidade de caracteres que a *string* contém.
- d. Se a frase digitada é igual ou não a "Eu estou estudando Java".
- e. A quantidade de palavras que a frase possui.

Atividade 03

^

Você acabou de conhecer o *ArrayList*, uma coleção de objetos que é muito mais prática de usar frente os *arrays* tradicionais. Armazene os dados de uma turma em uma coleção de **Alunos**. Um aluno tem nome, duas notas e uma média, que é calculada a partir dos valores das notas. Após receber os dados de vários alunos, enquanto o usuário desejar, apresente a média da turma e a quantidade de alunos informados.

Atividade 04

^

Um sindicato dos professores deseja um sistema que gerencie a contribuição sindical dos professores de uma cidade. Um professor possui nome, cpf, titulação (Doutor, Mestre, Graduado) e salário. Realize a entrada de dados para vários professores, enquanto o usuário desejar. Sabe-se que um Graduado contribui 1,2% do salário, já o Mestre 1,4% e o Doutor 1,6%. Ao final, após receber todos os dados dos professores, apresente o total arrecadado pelo sindicato e a quantidade de professores que são Mestre. Fique atento onde cada método e atributo estará distribuído na sua aplicação.

Atividade 05

^

Faça um programa Java que mostre os conceitos finais dos alunos de uma turma, enquanto o professor quiser informar dados dos alunos:

- a. Os dados de cada aluno (número de matrícula, alfanumérico, e nota, numérica) serão fornecidos pelo usuário.
- b. A tabela de conceitos segue a seguir:

Nota	Conceito
de 0,0 a 4,9	D
de 5,0 a 6,9	C
de 7,0 a 8,9	B
de 9,0 a 10,0	A

Atividade 06



Brasília é uma cidade que contém muitos triatletas amadores. A Federação precisa gerenciar a participação dos atletas nos eventos de triatlo. Cada atleta possui o número de inscrição (que deve ser único), nome, tempo de prova (em horas completas), idade e se o atleta é elite. Ele será considerado elite se o tempo de prova for menor que 5 horas. Um método `isElite` deve encapsular esse cálculo e retornará `true`, se for elite, e `false`, se não for. Sabe-se que participarão 1.200 atletas. Após receber os dados dos atletas, apresente o seguinte relatório:

- a. A Quantidade de atletas da ELITE.
- b. O melhor atleta da competição.
- c. Todos os atletas da categoria 30 a 34 anos.
- d. A média de tempo dos atletas.
- e. Todos os atletas que não são ELITES.

Você deve trabalhar com `ArrayList`.

Atividade 07



Receba do usuário uma frase e faça o que se pede baseado na API String:

- a. A quantidade de letras 'A' que a frase tem.
- b. O tamanho da frase (quantidade de caracteres).
- c. A qualificação da frase (menor que 10 caracteres – PEQUENA, maior que 9 e menor que 29 – MEDIA, maior que 30 é GRANDE).
- d. Apresente a frase informada em CAIXA ALTA.
- e. A quantidade de palavras que a frase contém.

Valide para verificar se a String contém a palavra "Corrupção", pois se não houver deve ser solicitada novamente ao usuário.

Saiba Mais

Para ampliar seu conhecimento a respeito desse assunto, veja abaixo a(s) sugestão(ões) do professor:

- Para conhecer um pouco mais sobre a imutabilidade dos Strings, faça a leitura da postagem [Java: Diferenças entre as classes String, StringBuffer e StringBuilder ↗](#), disponível no site Oficina da Net.

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.