# Unidade 3

O objetivo principal do Unidade 3 é apresentar a estrutura básica da Programação Orientada a Objetos, assim como seus conceitos teóricos detalhadamente. O aluno deve se familiarizar com conceitos de classes, objetos, métodos, construtores, encapsulamento, herança, interface, polimorfismo entre outros, assim como os padrões e convenções para tal. Este capítulo tem um denso embasamento teórico, então é muito importante que as aulas práticas tenham sempre relação com o conteúdo ministrado e para maior compreensão do aluno, é interessante que o nível de abstração seja desenvolvido utilizando-se de exemplos que tracem uma linha entre o mundo da programação e o mundo real, desta forma o aluno terá um entendimento melhor do capítulo e terá menos frustrações em seu aprendizado.

## Aula 1 - Classes, Objetos e Instâncias

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Entender a definição de Classes, Objetos, Instâncias e a técnica de encapsulamento

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Classes;
* Objetos;
* Instâncias;
* Encapsulamento.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 1.1 - Classes

Educador, o conceito de classe pode ser um dos conceitos mais frustrantes para o aprendizado do aluno. A melhor forma de se fixar esse conteúdo é reforçando-o a cada conceito ministrado e completando suas definições a cada aula, utilizando vários exemplos mais palpáveis, conforme dicas e exemplos do livro do aluno

#### 1.1.1 - Abstração

Abstração é a técnica de simplificar ou omitir detalhes que não são relevantes a aplicação. Uma técnica muito utilizada em POO.

Realize alguns exercícios de abstração, baseados no exemplo desta aula para fixar o conceito ao aluno.

#### 1.1.2 - Classes

Mostrar a importância da abstração na programação. É necessário que nossos programas tenham o mínimo de detalhes, que ele seja “limpo”, mas não confundir com organização. É sempre necessário que nossos programas sejam autoexplicativos, com seus comentários, padrões e convenções.

O exemplo das Blueprints é bastante interessante, você pode completar citando outros exemplos onde objetos são representados e certos atributos são omitidos por falta de utilidade na situação vigente.

### 1.2 - Objetos

A principal dúvida aqui é a diferença entre classe e objeto, já que ambos se assimilam. É interessante citar exemplos do nosso dia a dia, como o já mencionado exemplo do Carro, como também uma receita de bolo. Uma explicação básica é que um objeto tem um ciclo de vida, ele é inicializado, cumpre suas tarefas e é finalizado, as classes não.

Se a descrição acima não for o suficiente, sugiro a leitura das seguintes referências:

- Livros:

* Use a cabeça! Java, Sierra & Kathy – Alta Books
* Java: como programa, Harvey M. Deitel – Pearson

- Curso grátis:

* Code Academy: <https://www.codecademy.com/pt/learn/learn-java>

O curso sugerido acima é de nível básico. Se neste ponto os conceitos de Java e POO não estiverem sólidos, é recomendável este curso como reforço para o educador.

### 1.3 – Instâncias

Para exemplificar existem diversas analogias. Pense sempre que a classe será o “plano”, “projeto”, “modelo”, as “especificações”, as “regras” e o objeto/instância será sempre o “concreto”, o “palpável”, o que “diferencia”, a “coisa”, o “produto” e etc. Tente fazer exercícios de fixação com seus alunos. Pegue coisas do mundo real e tente definir se é objeto ou classe.

### 1.4 – Encapsulamento

O aluno não verá a necessidade de encapsular as classes pois, neste exemplo, poderíamos restringir a entrada do usuário e evitar as inconsistências acima. O grande problema é que a POO foi criada e desenvolvida também para facilitar o reuso e modularização dos componentes. O criador da classe saberá lidar com os possíveis erros, porém futuros usuários da classe talvez não percebam todas estas restrições e o programa apresentará erros de inconsistência e coesão de dados.

Oconceito de encapsulamento tem que ser entendido como um invólucro da classe. Uma camada a mais que garante a consistência dos dados. A medida que a classe vai se estendendo é possível adicionar mais destes invólucros na classe. Este invólucro trabalha como mediador entre o usuário da classe (neste sentido, o usuário é outra classe que faz uso desta em questão) e a classe em si. Por exemplo um controle remoto.

O invólucro seria os botões, eles abstraem os comandos do controle remoto e é o mediador entre a pessoa que faz uso dele e os componentes eletrônicos internos. Se não existissem os botões, teríamos que fazer ligações manuais entre os condutores do controle remoto para determinadas ações. Uma ligação errada destas poderia queimar o controle remoto. Esta é a importância do encapsulamento.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 1 > Exemplos > 1.4** no seu Netbeans.

### Exercícios

As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:

1. **O que você entende por abstração? Qual a relação com POO?**

Resposta: Abstração simplifica ou omite certos detalhes para reter apenas a informação enxuta. Quanto a relação com POO, classes são abstrações de objetos (ou "coisas") do mundo real.

1. **O que são Classes em POO? Explique detalhadamente.**

Resposta: Classes são como modelos que estão encarregadas de descrever as características relevantes de um elemento. Elas podem ter qualquer quantidade de métodos para acompanhar os níveis de acessibilidade de outros métodos. O uso de classes faz com que a estrutura do código seja mais clara e bem definida.

1. **O que são Objetos em POO? Explique detalhadamente.**

Resposta: Objetos são elementos que possuem estados e comportamentos. Uma vez definidos, os elementos podem existir com suas próprias características, pois isto garante a eles a oportunidade de agregar algum valor a um componente do programa sem que haja necessidade de uma estrutura muito extensa. Consequentemente, objetos são instâncias de classes.

1. **O que são Instâncias em POO? Explique detalhadamente.**

Resposta: Considere instância como uma outra maneira de dizer objeto. Um objeto que ocupa um espaço de memória é uma instância. Então para representar computacionalmente o objeto, usa-se o nome instância. Como visto nos capítulos anteriores, existem definições mais profundas para o que chamamos de instância. A principal é que ela é uma forma de diferenciar um objeto do outro.

1. **O que é encapsulamento? Explique com o exemplo da classe Pessoa.**

Resposta: É uma técnica utilizada para não expor os dados internos de uma classe para seu usuário, tornando partes de códigos mais independentes e modularizadas. O encapsulamento geralmente é utilizado para restringir ações das classes de modo que elas atendam a regra de negócio de uma maneira segura.

1. **O que é getter? E setter?**

Resposta: Os getters são utilizados para que um objeto acesse os valores dos atributos de um outro objeto. Os setters dão o poder de atribuir um valor a um atributo do objeto que possui este setter. Em resumo são obtentores e atribuidores de valor de um objeto.

1. **O que diferencia um Objeto de outro?**

Resposta: Uma instância. Mais precisamente o endereço de memória no qual o objeto está alocado.

1. **No cotidiano, o que pode ser comparado a uma Classe? E Objetos?**

Resposta: No cotidiano a classe é um Carro e o objeto é um Gol branco de 1997.

### 1.7 – TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

#### O Prazo

**Não há gente o suficiente para que o projeto seja finalizado dentro do prazo! Por enquanto todos os desenvolvedores estão alocados em outros projetos.**

**Um dos maiores problemas em um projeto de software. Um código profissional e com qualidade necessita de tempo e dedicação. Quando há um prazo apertado, geralmente contrata-se mais pessoas para desenvolver, porém as vezes um projeto não suporta tanta gente.**

**Quando há um excesso de desenvolvedores, os trabalhos podem ser conflitantes e a paralelização de atividades torna-se mais difícil.**

**Outra solução para atender um prazo apertado é diminuir a quantidade de artefatos.**

**Mas afinal o que são os artefatos?**

#### Artefatos

**Um artefato é qualquer documento ou algo que dê valor ao projeto. A próxima etapa seria a arquitetural. O arquiteto já identificou as classes, métodos e atributos e agora deveria colocá-las em um diagrama. Como estamos com o prazo apertado, vamos pular esta etapa e deixá-la para o final.**

**Para suprir a necessidade de mais desenvolvedores, você irá programar.**

#### Tarefa

**Implemente as classes identificadas. Teste todos os construtores, getters e setters.**

**Educador,** para não nos alongarmos demais e para que você possa praticar com melhor qualidade, iremos apresentar aqui apenas os passos que não são tão cruciais como, por exemplo, métodos e instruções que não foram ensinadas em aula ou que não constam na documentação oficial do Java.

Os passos executados nesta etapa foram:

1. Criação das classes
2. Inserção dos atributos de cada classe
3. Encapsulamento
4. Criação dos métodos adicionais da classe Movie.
5. A classe Movie tem alguns métodos especiais:
6. Formatação monetária:
   1. Primeiramente definimos o tipo dos campos revenue e budget sendo BigDecimal. Isto porque um BigDecimal pode ser um número bem grande mas possui apenas duas casas decimais.

**private** BigDecimal budget**;**

**private** BigDecimal revenue**;**

* 1. Foi criado um método auxiliar para converter o BigDecimal para o formato monetário em String.

**private** String **bigDecimalToString(**BigDecimal bigDecimal**)** **{**

*// Retornará sempre a moeda local*

NumberFormat numberFormat **=** NumberFormat**.**getCurrencyInstance**();**

**return** numberFormat**.**format**(**bigDecimal**);**

**}**

* 1. Também foi criado um método que faz o inverso (converte String para BigDecimal).

**private** BigDecimal **currencyStringToBigDecimal(**String string**)** **{**

**try** **{**

NumberFormat currencyFormat **=** NumberFormat**.**getCurrencyInstance**();**

Number number **=** currencyFormat**.**parse**(**string**);**

BigDecimal bigDecimail **=** **new** BigDecimal**(**number**.**toString**());**

**return** bigDecimail**;**

**}** **catch** **(**ParseException ex**)** **{**

**return** **null;**

**}**

**}**

* 1. Os métodos Setters de budget e revenue foram alterados para receber a String como parâmetro e convertê-la para BigDecimal utilizando o método criado no item 1.3, acima.

**public** **void** **setBudget(**String budget**)** **{**

**this.**budget **=** currencyStringToBigDecimal**(**budget**);**

**}**

**public** **void** **setRevenue(**String revenue**)** **{**

**this.**revenue **=** currencyStringToBigDecimal**(**revenue**);**

**}**

* 1. Um método getter foi adicionado para budget e revenue. Este retorna uma String em formato monetário, utilizando o método criado no item 1.2 acima.

**public** String **getFormattedBudget()** **{**

**return** bigDecimalToString**(**budget**);**

**}**

**public** String **getFormattedRevenue()** **{**

**return** bigDecimalToString**(**revenue**);**

**}**

1. Formatação de datas:
   1. Modificamos o setter do atributo releaseDate que recebe uma data em String e converte para o tipo Date.

**public** **void** **setReleaseDate(**String releaseDate**)** **{**

SimpleDateFormat sdf **=** **new** SimpleDateFormat**(**"dd/MM/yyyy"**);**

**try** **{**

**this.**releaseDate **=** sdf**.**parse**(**releaseDate**);**

**}** **catch** **(**ParseException ex**)** **{**

**this.**releaseDate **=** **null;**

**}**

**}**

Veja a documentação de try catch e exceptions

* 1. Criamos um getter para o atributo releaseDate que retorna a data em formato de String.

**public** String **getFormattedReleaseDate()** **{**

SimpleDateFormat format **=** **new** SimpleDateFormat**(**"dd/MM/yyyy"**);**

**return** format**.**format**(**releaseDate**);**

**}**

1. Formatação da duração do filme
   1. Definimos que o tipo de duration será long. É o tipo mais apropriado para medição de tempo.
   2. Criamos um getter para exibir a duração do filme de uma maneira formatada. Ex.: 110 min

**public** String **getFormattedDuration()** **{**

**return** duration **+** " min"**;**

**}**

1. Controle e manipulação do team e cast serão implementados nas aulas seguintes.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 1 > TDP** no seu Netbeans.

### COTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

* Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall.

Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Nesta aula abordamos algumas definições gerais sobre Orientação a Objetos. Neste livro podemos encontrar mais detalhes como conteúdo complementar.

* Leia a obra **Java – The Ultimate Beginner’s Guide!,** do autor Andrew Johansen.

Este livro apresenta: um conteúdo resumido para iniciantes em Java. Apesar de estar no idioma inglês, este livro é mais objetivo e resumido que a obra do Deitel. Em pouco mais de vinte páginas, podemos ter uma visão geral sobre Java e Orientação a Objetos.

### REFERÊNCIAS

* JOHANSEN, ANDREW.  **Java – The Ultimate Beginner’s Guide!**. Publicação independente.  Kindle Edition, 2015
* DEITEL.  **Java – Como Programar**. Prentice Hall.  8a Edição, 2010

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno

* WIKIPEDIA. **Abstraction (Software Engineering)**. Disponível em: < https://en.wikipedia.org/wiki/Abstraction\_(software\_engineering) >. Acesso em: 13 de. 2016.
* SIERRA & KATHY. **Use a cabeça! Java**. Alta Books. 2a Edição, 2005.
* CODE ACADEMY. **Learn Java.** Disponível em: <https://www.codecademy.com/pt/learn/learn-java>. Acesso em: 14 dec. 2016.

## Aula 2 - Herança e Interface

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para:

* Entender conceitos mais aprofundados da POO;
* Implementar uma herança;
* Implementar uma interface.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Introdução;
* Herança;
* Herança múltipla;
* Interfaces.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 2.1 – Introdução

**Educador, nessa aula é importante que você explique que e**xistem diversas convenções e padrões de projeto para programação orientada a objetos. Esses padrões são definidos por grupos de entusiastas da área e são adotados pela comunidade à medida que eles ganham popularidade.

### 2.2 – Herança

Informe que herança se trata de uma relação hierárquica entre classes. Quando queremos que uma classe compartilhe seus métodos e atributos com a outra criamos a relação de herança, em que a classe com recursos compartilhados é a superclasse ou classe mãe, e a classe que recebe estes recursos é chamada de subclasse ou filha.

#### 2.2.1 - A classe Object

Agora, explique que a classe Object é a classe mãe de todas as classes e que ela possui alguns métodos e atributos que automaticamente fazem parte de todas as outras classes do Java.

Para saber mais, consulte a documentação oficial da classe Object (<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>)

### 2.3 – Herança Múltipla

Educador, fale sobre herança múltipla, para ajuda-lo segue a resposta do site Stackoverflow sobre esse tema: Em Java não é possível que uma classe faça herança de múltiplas classes. O motivo disso, imagine que uma classe herda outras duas classes, entretanto essas duas classes herdadas possuem métodos com a mesma assinatura, porém com implementações diferentes. Seria algo difícil de lidar, pois como a subclasse saberia qual implementação do método ela deveria usar? Por isso não é possível que uma classe estenda de mais de uma classe em Java.

Uma classe abstrata pode possuir métodos abstratos assim como métodos concretos, ou seja, métodos que já foram implementados, então o fato da classe ser abstrata não ajuda muito a resolver o problema citado anteriormente.

O que se pode fazer em Java é que uma classe implemente várias interfaces, pois interfaces são como classes 100% abstratas, ou seja, é completamente proibido qualquer tipo de implementação nela, logo, não há o menor problema se as duas interfaces tiverem um método com a mesma assinatura, pois quem vai implementar o método é a classe que implementa as interfaces.

Algo curioso é que a palavra reservada extends pode ser usada para as interfaces, que é quando uma interface herda outra (s) interface (s), mas nesse caso não há problema de uma interface herdar de várias outras interfaces, pois não há nenhuma implementação envolvida nesse caso.

(Fonte: Java não possui herança múltipla?, diz stackoverflow. Disponível em: <<http://pt.stackoverflow.com/a/22720>>. Acesso em 13 dez. 2016.)

Como mencionado no livro do aluno, o C++ é uma das linguagens que suportam herança múltipla. C++ é uma linguagem mais antiga e hoje é utilizada para aplicações de mais baixo nível, que demandam uma otimização de código e processamento maiores, como por exemplo, um jogo de última geração.

### 2.4 – Interfaces

A nomenclatura das interfaces é derivada do inglês e foi amplamente adotada pela comunidade de programadores. O sentido de ser **domesticável** e não **doméstico** é devido aos níveis de abstração existentes na linguagem de programação. Para que os alunos tenham um melhor entendimento, exemplifique! A seguir uma exemplificação: Com um nível de abstração mais detalhado podemos dizer que um gato é doméstico, mas, e, se não quisermos que todos os gatos sejam domésticos?

A interface nos dá a possibilidade de instanciarmos gatos domésticos e não doméstico e semanticamente, se o gato pode ser ou exercer coisas ou não, então a nomenclatura seria domesticável, e não doméstico.

### Exercícios

As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:

1. **O que é herança em POO?**

Resposta: Herança é a predisposição de um objeto herdar características de outro.

**2. Como especificar uma herança Java?**

Resposta: Na declaração da classe utilize a palavra reservada "extends" e após isso especifique a classe herdada.

**3. O que é classe Object?**

Resposta: A classe Object é a classe base do Java. Nela contém todo conteúdo básico de um objeto, como os métodos toString(), equals() e etc. Esta classe não possui nenhuma classe mãe, então podemos dizer que é a classe mais primitiva dentre as classes do Java.

**4. Qual é a classe que todas as outras classes estendem?**

Resposta: A classe que todas as outras estendem é a Classe Object.

**5. O que é herança múltipla? Por que não é utilizada em Java?**

Resposta: Herança múltipla é o ato de uma classe herdar características de duas ou mais classes. Por ser difícil manter o controle das informações e da semântica da estrutura de herança ela não é utilizada em Java.

**6. O que é interface?**

Resposta: Interface é o recurso que obriga um determinado grupo de classes a seguirem um contexto em comum.

**7. Dê um exemplo, diferente do livro, de herança.**

Resposta: Homem herda de humano e mulher herda de humano.

**8. Dê um exemplo, diferente do livro, de interface.**

Resposta: Um outro exemplo de interface é a Serializable da própria API Java. O Serializable é um contrato que a classe que a implementa assina para que ela possa ser serializavél e deserializável, isto é, transformam as classes em um pedaço de código binário legível por outras aplicações.

### TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

**Nelson já voltou da licença médica e você voltou para o cargo de programador.**

**Seu desempenho como gerente de projetos foi ótimo. E ao final do mês receberá uma bonificação no salário.**

#### 2.7.1 - Desenvolvimento

**Você agora é oficialmente programador Java da empresa. Todos os outros programadores preferem usar linguagem procedural. Foi difícil convencê-los a usar Java.**

**Neste projeto você vai ajudá-los a firmar a ideia de que a POO é melhor do que a programação procedural.**

#### Tarefa

**Identifique e implemente heranças e interfaces no código, se necessário**

### SUGESTÃO:

Para a implementação das heranças e interfaces observe que **Membro do elenco** É UM **Ator** e um **Ator** É UM **Membro da equipe.**  Então faz sentido fazermos esta extensão.

Observe também a alta frequência do atributo name e este será generalizado em uma classe mãe. Analisando o sentido semântico de todas as classes que possuem este atributo, não conseguimos encontrar um nome faria sentido semanticamente para todas as classes que estenderiam esta classe mãe, então o nome será GenericModel.

Outra observação importante, é que na hora de implementarmos as Views e Controllers devemos levar em consideração os models que irão popular as tabelas. Definindo uma interface, conseguimos generalizar estes models de forma que as AbstractTableModels consigam definir qual será o conteúdo da primeira e segunda coluna das respectivas tabelas. Vamos primeiro implementar a estrutura de Atores, Elenco e Equipe.

1. Na classe Actor, estenda de TeamMember.
2. Agora na classe CastMember, estenda de Actor.
3. Remova os atributos replicados (incluindo seus getters e setters) nas classes filhas e netas de TeamMember (name e position, por exemplo).
4. A diferença de um ator e um membro da equipe é que o cargo do ator (position) sempre será Ator e que ele pode ser um membro do elenco. Implemente esta restrição na classe Actor:

**public** **class** **Actor** **extends** TeamMember **{**

**public** **Actor(**String name**)** **{**

**super(**name**);**

**this.**name **=** name**;**

**this.**position **=** Position**.**Actor**;**

**}**

*// Bloqueia o uso do setPosition()*

@Override

**public** **void** **setPosition(**Position position**)** **{**

**throw** **new** UnsupportedOperationException**();**

**}**

@Override

**public** String **toString()** **{**

**return** "Ator: " **+** name**;**

**}**

**}**

1. Em segundo, implemente a estrutura de GenericModel (por enquanto ela apenas evita que todos os models precisem implementar o atributo name, mas logo ela fará mais sentido).
2. Crie a classe GenericModel, insira o atributo name e o encapsule.
3. Estenda de GenericModel todas as classes que possuem o atributo name.
4. Remova o atributo name (incluindo seus getters e setters) das classes filhas de GenericModel.

Por fim, vamos deixar nossas classes flexíveis para serem utilizadas nas tabelas da nossa aplicação:

1. Crie a interface TableModel
2. Defina dois métodos a ela: getFirstColumn() e getSecondColumn() que irão se responsabilizar por mostrar o dado na primeira e segunda coluna da tabela, respectivamente:

**public** **interface** **TableModel** **{**

**public** String **getSecondColumn();**

**public** String **getFirstColumn();**

**}**

1. Praticamente todos os nossos models serão recurso para preenchimento de alguma tabela então implemente a interface na classe GenericModel.
2. O método getFirstColumn() sempre retornará o valor do atributo name.
3. O método getSecondColumn() retornará um valor a critério do model, então sobrescreva este método em cada model que estende a classe GenericModel.
4. A classe GenericModel ficará da seguinte maneira:

**public** **abstract** **class** **GenericModel** **implements** TableModel **{**

**protected** String name**;**

**public** String **getName()** **{**

**return** name**;**

**}**

**public** **void** **setName(**String name**)** **{**

**this.**name **=** name**;**

**}**

@Override

**public** String **getFirstColumn()** **{**

**return** name**;**

**}**

@Override

**public** String **getSecondColumn()** **{**

**return** **null;**

**}**

@Override

**public** String **toString()** **{**

**return** name**;**

**}**

**}**

1. Confira a sugestão de qual deverá ser o retorno do método getSecondColumn() de cada model no material de apoio, mas fica critério do desenvolvedor escolher qual será este retorno.

A interface TableModel será amplamente utilizada nos passos seguintes e você poderá entender melhor a importância da mesma.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 2 > TDP** no seu Netbeans.

### COTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

* Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall.

Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Todo conceito sobre heranças e interfaces estão contidos em detalhes neste livro.

* Leia a 5a edição (2006) da obra **C++ – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall.

Este livro apresenta: aulas desde básicas até avançadas sobre programação C++. Este livro é extenso igual a versão sobre Java do Deitel, mas para nível de comparação do Java com outra linguagem de programação OO (C++) vale a pena a leitura.

### REFERÊNCIAS

* JOHANSEN, ANDREW.  **Java – The Ultimate Beginner’s Guide!**. Publicação independente.  Kindle Edition, 2015

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno

* STACK OVERFLOW. **Java não possui herança múltipla?**. Disponível em: <http://pt.stackoverflow.com/questions/22718/java-n%C3%A3o-possui-heran%C3%A7a-m%C3%BAltipla>. Acesso em: 13 dec. 2016.
* ORACLE. **Object.** Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>. Acesso em: 14 dec. 2016

## Aula 3 - Polimorfismo

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para:

* Compreender assuntos aprofundados da orientação a objetos
* Dominar o conceito de polimorfismo

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 50 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 20 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Polimorfismo;
* Métodos estáticos;
* Classes e métodos abstratos.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 3.1 – Polimorfismo

Polimorfismo é o ato de um objeto assumir diferentes formas. Educador, no exemplo dado no livro do aluno, é possível elucidar os conceitos de polimorfismo e ainda relacioná-los aos de herança. Provavelmente o aluno deverá ter dúvidas sobre privacidade, nomenclaturas e conceitos já aprendidos, nesses casos retome os pontos importantes dos capítulos anteriores.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 3 > Exercícios > 3.1** no seu Netbeans.

#### 3.1.1 – Sobreposição e sobrecarga

Educador, ensine que quando se estende uma classe, existe a possibilidade de reescrever os métodos da superclasse para adequar às condições da subclasse. Isso é chamado de sobreposição. A sobrecarga permite que existam vários métodos com o mesmo nome, desde que eles possuam assinaturas (quantidade e tipos de parâmetros) diferentes.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 3 > Exercícios > 3.1.1** no seu Netbeans.

### 3.2 – Métodos estáticos

Informe ao aluno que métodos estáticos são métodos cuja classe não precisa ser instanciada para ser utilizado, você pode usar como exemplo a classe Math.

### 3.3 – Classes e métodos abstratos

Educador, explique aos alunos que em POO pode-se abstrair classes e métodos. Métodos abstratos são métodos que não podem ser utilizados se uma subclasse não o implementar e classes abstratas são classes que não podem ser instanciadas, pois como seu nome já diz, são abstratas ou simplificadas.

#### 3.3.1 - Classes abstratas

Informe que classes abstratas são classes que não podem ser instanciadas, pois como seu nome já diz, são abstratas (ou simplificadas).

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 3 > Exercícios > 3.3.1** no seu Netbeans.

#### 3.3.2 – Métodos abstratos

Informe, também, que métodos abstratos são métodos que não podem ser utilizados se uma subclasse não o implementar.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 3 > Exercícios > 3.3.2** no seu Netbeans.

### Exercícios

As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:

1. **O que é polimorfismo em programação?**

Resposta: Em programação trata-se da capacidade um objeto ser referenciado de diversas formas. O objeto sempre vai ser do mesmo tipo, o que muda é a maneira como nos referimos a ele.

1. **O que é sobreposição e sobrecarga?**

Resposta: Quando estendemos uma classe, temos a possibilidade de reescrever os métodos da superclasse para adequar às condições da subclasse. Isto é chamado de sobreposição. A sobrecarga permite que exista vários métodos com o mesmo nome, desde que eles possuam uma assinatura (quantidade e tipos de parâmetros) diferentes.

1. **O que são métodos estáticos?**

Resposta: Métodos estáticos independem das variáveis de instância (atributos) de uma classe, e executam numa função sem dependência do conteúdo do objeto. Digamos que quando um método não altera o estado do objeto, ele pode ser transformado para um método estático.

1. **O que são classes abstratas?**

Resposta: As classes abstratas são classes não instanciáveis. Somente as suas subclasses poderão ser instanciadas.

1. **O que é um método abstrato?**

Resposta: Um método abstrato é similar a classe abstrata, ele não pode ser executado, a não ser que foi sobreposto em uma subclasse. O método abstrato poderá estar em uma classe não abstrata, porém poderá ser executado apenas se a subclasse o implementar.

1. **Qual a relação entre Polimorfismo e Sobrecarga?**

Resposta: Sobrecarga é uma técnica do Polimorfismo que permite que criemos diversos métodos com mesmos nomes, porém assinatura (parâmetros) diferentes em uma mesma classe ou em suas subclasses.

1. **O que acontece se instanciarmos um objeto de uma classe abstrata?**

Resposta: Como se trata de uma classe abstrata, um erro acontecerá e o código não será executado, pois não é possível instanciar uma classe abstrata. É possível instanciar apenas classes concretas.

1. **O que acontece se tentarmos usar um método abstrato?**

Resposta: Um erro acontecerá e o código não será executado. Se este método não estiver sido sobrescrito nas subclasses deste método, este erro acontecerá, pois se trata de um método não implementado. Não poderíamos dizer nem que o método é vazio, ele é simplesmente nulo.

### TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

**Os desenvolvedores estão gostando da ideia de utilizar POO. Para convencê-los de vez e deixá-los de queixo caído, mostre o poder do Polimorfismo.**

#### Tarefa

**Abstraia mais e tente identificar possíveis usos de polimorfismo e classes estáticas. Implemente.**

A classe GenericModel pode ser do tipo abstrato. Nos passos futuros você poderá identificar e implementar mais classes abstratas e métodos estáticos se assim for necessário. A implementação do enum Position pode ser feita neste momento. Na unidade 5 iremos utilizar o padrão de projeto Singleton. Padrões de projetos são estratégias para reaproveitar soluções de problemas comuns de desenvolvimento. Estude este padrão para que quando chegarmos lá isto não seja um fator bloqueante.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 3 > Aula 3 > TDP** no seu Netbeans.

### CONTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

* Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall.

Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Todo conceito sobre heranças e interfaces estão contidos em detalhes neste livro.

### REFERÊNCIAS

* JOHANSEN, ANDREW.  **Java – The Ultimate Beginner’s Guide!**. Publicação independente.  Kindle Edition, 2015
* PRESSMAN, ROGER S. **Engenharia de Software – Uma Abordagem Profissional.** ARTMED. 7a Edição, 2011.

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno

* ORACLE. **Java™ Platform, Standard Edition 8 API Specification**. Disponível em: <https://docs.oracle.com/javase/8/docs/api/overview-summary.html>. Acesso em: 13 dec. 2016.
* ORACLE. **Access Control**. Disponível em: <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>. Acesso em: 13 de. 2016.
* CAELUM. **Herança, reescrita e polimorfismo**. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/>. Acesso em: 13 dec. 2016.
* FREEMAN, ERIC; FREEMAN, ELISABETH. **Use a cabeça! Padrões de Projetos**. Alta Books. 2a Edição, 2007.
* GAMMA, ERIC; JOHNSON, RALPH; HELM, RICHARD; VLISSIDES, JOHN. **Padrões de Projeto - Soluções Reutilizaveis de Software Orientado a Objetos**. Addison-Wesley. 1a Edição, 1994.