Projetando classes

Apresentação

Em programação e na orientação a objetos, existem elementos importantes que refletem no comportamento dos objetos. A classe é um tipo abstrato de dados (TAD), utilizada para abstrair um conjunto de objetos que sejam representados por características similares.

Para desenvolvermos aplicações utilizando o paradigma de orientação a objetos, podemos definir que projetar classes é um importante desafio. Segundo Horstmann (2009), classes são coleções de objetos; portanto, precisamos começar a atividade de programação identificando os objetos e as classes às quais eles pertencem.

Nesta Unidade de Aprendizagem, você vai estudar os conceitos de coesão e acoplamento, que são importantes para formar uma estrutura de classes consistente. Além disso, vai estudar o conceito de pacotes, que são uma forma de organizar as classes do nosso projeto.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Identificar objetos e definir a quais classes eles pertencem.
- Definir coesão e acoplamento.
- Usar pacotes para organizar classes.

Desafio

Projetar uma classe pode ser um bom desafio, mas, quando seguimos algumas regras e padrões, nosso trabalho se torna menos árduo.

Analise o seguinte cenário:

Você trabalha como analista/programador em uma fábrica de *softwares*. Seu gerente solicitou que você atuasse em um projeto para o novo cliente, criando uma aplicação eficiente para atender às necessidades de controle de uma oficina mecânica. O principal objetivo é projetar uma aplicação capaz de:

- automatizar o processo de manutenção de veículos;
- realizar o cadastro de clientes;
- controlar a entrada de veículos no setor de manutenção.

O módulo do sistema desenvolvido para cadastro de cliente precisa atender aos seguintes requisitos:

- CPF;
- nome do cliente;
- endereço do cliente;
- telefone do cliente;
- *e-mail* do cliente.

O módulo do sistema desenvolvido para cadastro de veículos precisa atender aos seguintes requisitos de armazenamento de dados:

- placa do veículo;
- modelo do veículo;
- ano do veículo;
- fabricante do veículo;
- cor do veículo.

Para executar essa tarefa, você precisará fazer o seguinte:

- Criar um pacote chamado "modelo" – no qual serão armazenadas as classes “modelo”.
- Criar um pacote chamado "controle" – no qual será armazenado o método *main*.

A classe deve ser nomeada como “Principal” e será utilizada para instanciar as classes modelo e iniciar o programa. Seu gerente sugeriu o uso de uma linguagem de programação orientada a objetos (POO). Também sugeriu o uso da plataforma Java, utilizando o IDE NetBeans. Você precisa encaminhar o arquivo em formato .zip até o final do dia para oficializar a execução do projeto, seguindo todas as instruções.

Infográfico

Os conceitos básicos sobre classes devem ser conhecidos pelos desenvolvedores que desejam desenvolver com base em linguagens orientadas a objetos. Entre os princípios que evitam problemas no uso de classes estão a coesão e o acoplamento. Ambos estão relacionados com os módulos de um sistema e associados aos conceitos de classes e à sua aplicação à POO.
Enquanto a coesão é o grau de responsabilidade única bem definida para uma classe, o acoplamento é o grau com o qual um módulo define sua dependência em relação aos demais, para garantir sua funcionalidade.

Veja, no Infográfico, uma descrição sobre escolha de classes e os princípios de coesão e acoplamento.

OS PRINCÍPIOS DAS CLASSES

Assim como todos os componentes da programação orientada a objetos, as classes também são construídas a partir de alguns princípios. Vamos conhecer as características de cada um deles no infográfico a seguir.

A ESCOLHA DE CLASSES

Conceitos importantes

- As classes são coleções de objetos, e seus nomes devem ser substantivos.
- Os princípios do desenvolvimento de *software* que as classes implementam são: coesão, acoplamento e pacote.

COESÃO

Coesão é o princípio que define a responsabilidade única e defende que uma classe deve assumir apenas uma responsabilidade, sem se apossar daquilo que não é seu dever dentro do código ou programa. Uma boa classe deve representar um único conceito.

Quando esse princípio não é respeitado, o código passa a representar problemas principalmente ligados à sua manutenção.

A seguir, acompanhe um exemplo de código fora do padrão de coesão.

COMO APLICAR?

```
public class CadastroDeProdutos
 {
     public void ExibirFormulario()
{
         //implementação
     }

     public void ObterProduto()    {
         //implementação
     }

     public void gravarProdutoDB   {
         //implementação
     }
}
```

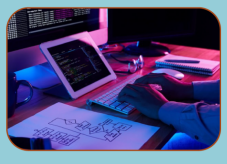
Note que a classe tem mais de uma responsabilidade. Agora, veja o próximo código, no qual a classe responsável pelo cadastro não assume as demais responsabilidades e chama a função adequada para que os dados do produto cadastrado sejam finalizados. Seu papel é somente tratar do formulário.

```
public class Programa
 {
       public void MostrarFormulario()
{
          //Implementação
       }

     public void BotaoGravarProduto( )
{
          Produto.gravarProduto();
     }

 }
```

ACOPLAMENTO

O acoplamento também é um princípio importante que define em que situações as classes dependem umas das outras. Basicamente, quanto maior for essa dependência entre as classes vinculadas entre si, maior é o acoplamento, e diz-se que as classes estão fortemente acopladas. Assim como a falta de coesão, o acoplamento pode trazer problemas. Observe um código nessa situação:

```
public class X
 {
     private ClasseConcretaY var1;

     void M1(ClasseConcretaW var2 ) {
… }
 }
```



O forte acoplamento:

- torna mais difícil compreender as classes isoladamente;
- forma mudanças quando as classes relacionadas sofrem alterações.

Para evitar tais problemas, deve-se programar para uma interface, e não para uma implementação.

As boas práticas aplicadas aos projetos orientados a objetos devem ser consideradas, porque são extremamente importantes para garantir a funcionalidade de classes e a sua hierarquia. Utilizar os recursos e aplicar os princípios corretamente pode evitar diversos problemas, que consequentemente afetariam a entrega do programa e o resultado esperado.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Conteúdo do Livro

Escolher, nomear e estruturar bem as classes de uma aplicação, assim como organizá-las em pacotes, é de extrema importância no paradigma de orientação a objetos. Portanto, o domínio desses conceitos se torna vital para o desenvolvimento de *softwares*.

Para se aprofundar sobre essas questões, leia o trecho da obra **Conceitos de computação com Java**, base teórica desta Unidade de Aprendizagem.

Boa leitura.

CAY HORSTMANN

CONCEITOS DE COMPUTAÇÃO COM

Compatível com
Java 5 & 6

Java

5ª Edição

bookman®

Sobre o autor

CAY S. HORSTMANN é professor de ciência da computação no departamento de ciência da computação da Universidade Estadual de San Jose. É experiente programador, foi vice-presidente e gerente de tecnologia da Preview Systems, Inc. Presta consultoria em C++, Java, Windows e programação Internet para importantes corporações, universidades e organizações. Horstmann é autor de muitos livros profissionais e acadêmicos bem-sucedidos, incluindo *Padrões e Projetos Orientados a Objetos*, *Big Java*, *Conceitos de Computação com o Essencial de C++* (publicados pela Bookman Editora sob esses títulos), *Big C++* e *Core Java*, com Gary Cornell.

H819c Horstmann, Cay.
Conceitos de computação em Java [recurso eletrônico] / Cay Horstmann ; tradução Edson Furmankiewicz. – 5. ed. – Dados eletrônicos – Porto Alegre : Bookman, 2009.

Editado também como livro eletrônico em 2009.
Conteúdo: Capítulos 16, 17, 18 e apêndices de D a M disponíveis em: www.bookman.com.br.
ISBN 978-85-7780-407-8

1. Computação – Linguagem de programação. I. Título.

CDU 004.438JAVA

Catalogação na publicação: Renata de Souza Borges CRB-10/Prov-021/08

8.1 Escolhendo classes

Você já utilizou um bom número de classes nos capítulos anteriores e provavelmente projetou algumas como parte de seus exercícios de programação. Projetar uma classe pode ser um desafio – nem sempre é fácil dizer como começar ou se o resultado é de boa qualidade.

Estudantes com alguma experiência anterior em programação, em uma outra linguagem, estão acostumados a programar *funções*. Uma função executa uma ação. Na programação orientada a objetos, as ações aparecem como métodos. Cada método, porém, pertence a uma classe. Classes são coleções de objetos e objetos não são ações – eles são entidades. Portanto, você precisa começar a atividade de programação identificando objetos e as classes às quais eles pertencem.

Lembre-se da regra geral do Capítulo 2: nomes de classe devem ser substantivos e nomes de método devem ser verbos.

Uma classe deve representar um único conceito do domínio do problema, como negócios, ciência ou matemática.

O que faz uma boa classe? Acima de tudo, uma classe deve *representar um único conceito.* Algumas classes que vimos representam conceitos matemáticos:

- `Point`
- `Rectangle`
- `Ellipse`

Outras classes são abstrações de situações da vida real.

- `BankAccount`
- `CashRegister`

Para essas classes, as propriedades de um objeto típico são fáceis de entender. Um objeto `Rectangle` tem uma largura e uma altura. Dado um objeto `BankAccount`, você pode depositar e sacar dinheiro. Geralmente, conceitos da parte do universo compreendida pelo programa, como ciências, negócios ou um jogo, fazem boas classes. O nome para essa classe deve ser um substantivo que descreve o conceito. Alguns nomes de classes padrão em Java são um pouco estranhos, como `Ellipse2D.Double`, mas você pode escolher nomes mais apropriados para suas classes.

Uma outra categoria útil de classe pode ser descrita como *atores*. Objetos de uma classe de atores realizam alguns trabalhos para você. Exemplos de atores são a classe `Scanner` do Capítulo 4 e a classe `Random` no Capítulo 6. Um objeto `Scanner` varre um fluxo de dados em busca de números e strings. Um objeto `Random` gera números aleatórios. Se utilizar a língua inglesa para atribuir nomes, uma boa idéia é escolher nomes de classes para os atores que terminem em "er" ou "or". (Um nome melhor para a classe `Random` poderia ser `RandomNumberGenerator`.)

Ocasionalmente, uma classe não tem nenhum objeto, mas contém uma coleção de constantes e métodos estáticos relacionados. A classe `Math` é um exemplo típico. Uma classe assim é chamada de *classe utilitária*.

Por fim, você viu classes com somente um método `main`. O único propósito dessas classes é iniciar um programa. Da perspectiva de projeto, estes são exemplos mais ou menos degenerados de classes.

O que não poderia ser uma boa classe? Se você não pode afirmar a partir do nome da classe o que um objeto da classe supostamente deve fazer, provavelmente você não está no caminho certo. Por exemplo, sua lição de casa poderia solicitar para você escrever um programa que imprima cheques de pagamento. Suponha que inicialmente você tente projetar uma classe `PaycheckProgram`. O que um objeto dessa classe faria? Um objeto dessa classe teria que fazer tudo o que a lição de casa exige fazer. Isso não simplifica nada. Uma classe melhor seria `Paycheck`. Seu programa pode então manipular um ou mais objetos da classe `Paycheck`.

Um outro equívoco comum, especialmente de estudantes habituados a escrever programas que consistem em funções, é transformar uma ação em uma classe. Por exemplo, se sua lição de casa fosse calcular um cheque de pagamento, você poderia considerar escrever uma classe `ComputePaycheck`. Mas você consegue visualizar um objeto do tipo ComputePaycheck? O fato de que ComputePaycheck não é um substantivo leva você a achar que está no caminho errado. Por outro lado, uma classe `Paycheck` tem um sentido intuitivo. O termo "cheque de pagamento" (ou Paycheck) é um substantivo. Você pode visualizar um objeto de cheque de pagamento. Então você pode pensar em métodos úteis para a classe `Paycheck`, como `computeTaxes`, que o ajudarão a resolver a lição de casa.

AUTOVERIFICAÇÃO DA APRENDIZAGEM

1. Qual é a regra geral para escolher classes?
2. Seu trabalho é escrever um programa de xadrez. A classe ChessBoard (`Tabuleiro`) poderia ser uma classe apropriada? Que tal `MovePiece`?

8.2 Coesão e acoplamento

Nesta seção você aprenderá dois critérios úteis para analisar a qualidade da interface pública de uma classe.

A interface pública de uma classe é coesa se todos os seus recursos estiverem relacionados ao conceito que a classe representa.

Uma classe deve representar um único conceito. As constantes e os métodos públicos que a interface pública expõe devem ser *coesos*. Isto é, todos os recursos da interface devem estar intimamente relacionados ao único conceito que a classe representa.

Se achar que a interface pública de uma classe referencia vários conceitos, isso é um bom sinal de que pode ser a hora de utilizar classes separadas. Considere, por exemplo, a interface pública da classe `CashRegister` no Capítulo 4:

```
public class CashRegister
{
   public void enterPayment(int dollars, int quarters,
         int dimes, int nickels, int pennies)
   . . .
   public static final double NICKEL_VALUE = 0.05;
   public static final double DIME_VALUE = 0.1;
   public static final double QUARTER_VALUE = 0.25;
   . . .
}
```

Na verdade há dois conceitos aqui: uma caixa registradora que armazena moedas e calcula o total e os valores das moedas individuais. (Por simplicidade, supomos que a caixa registradora só contém moedas, não cédulas. O Exercício P8.1 discute uma solução mais geral.)

Faz sentido ter uma classe `Coin` separada e tornar as moedas responsáveis pelos seus valores.

```
public class Coin
{
   public Coin(double aValue, String aName) { . . . }
   public double getValue() { . . . }
   . . .
}
```

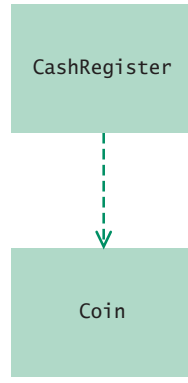


Figura 1
Relacionamento de dependência entre as classes `CashRegister` e `Coin`.

A classe `CashRegister` pode então ser simplificada:

```
public class CashRegister
{
   public void enterPayment(int coinCount, Coin coinType) { . . . }
   . . .
}
```

Agora a classe `CashRegister` não precisa conhecer mais nada sobre os valores das moedas. A mesma classe pode muito bem tratar euros ou zorkmids!

Claramente essa é uma solução melhor, porque ela separa as responsabilidades entre a caixa registradora e as moedas. A única razão por que não seguimos essa abordagem no Capítulo 4 foi manter o exemplo da `CashRegister` simples.

Uma classe depende de outra classe se ela usa objetos dessa classe.

Muitas classes precisam de outras classes para que possam fazer o seu trabalho. Por exemplo, a classe `CashRegister` reestruturada agora depende da classe `Coin` para determinar o valor do pagamento.

Para visualizar os relacionamentos, como a dependência entre as classes, programadores criam diagramas de classes. Neste livro, utilizamos a notação UML ("Unified Modeling Language") para objetos e classes. UML é uma notação para análise orientada a objetos cujo projeto foi criado por Grady Booch, Ivar Jacobson e James Rumbaugh, três importantes pesquisadores em desenvolvimento de software orientado a objetos. A notação UML distingue entre *diagramas de objetos* e diagramas de classes. Em um diagrama de objetos os nomes das classes são sublinhados; em um diagrama de classes os nomes das classes não são sublinhados. Em um diagrama de classes, você indica a dependência por meio de uma linha tracejada usando uma seta com a ponta aberta apontando para a classe dependente. A Figura 1 mostra um diagrama de classes que indica que a classe `CashRegister` depende da classe `Coin`.

Observe que a classe `Coin` *não* depende da classe `CashRegister`. Moedas não fazem idéia de que são coletadas pela caixa registradora e elas podem realizar suas funções sem nunca chamar um método qualquer na classe `CashRegister`.

Se muitas classes de um programa dependerem umas das outras, dizemos então que o *acoplamento* entre as classes é alto. Inversamente, se houver poucas dependências entre as classes, dizemos que o acoplamento é baixo (veja Figura 2).

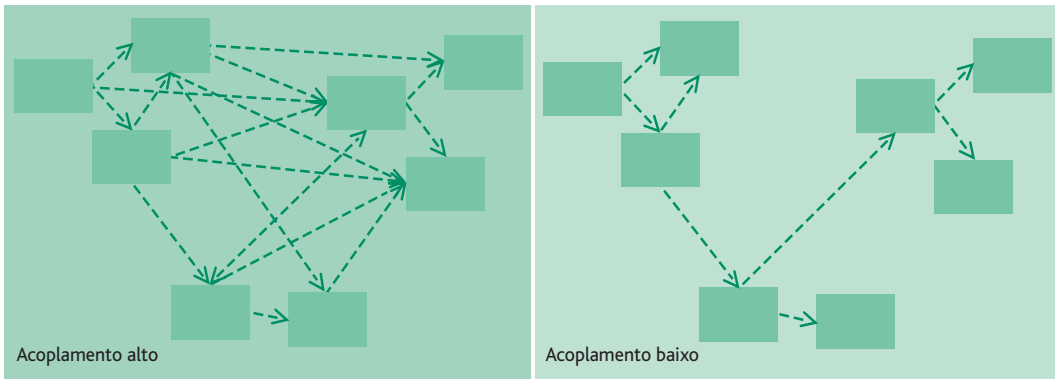


Figura 2 Acoplamento alto e acoplamento baixo entre classes.

Uma boa prática é minimizar o acoplamento (isto é, a dependência) entre classes.

Por que o acoplamento é importante? Se a classe `Coin` muda na próxima distribuição do programa, todas as classes que dependem dela podem ser afetadas. Se a alteração for drástica, todas as classes associadas devem ser atualizadas. Além disso, se quisermos utilizar uma classe em outro programa, teremos que levar junto todas as classes das quais ela depende. Portanto, queremos remover o acoplamento desnecessário entre as classes.

Autoverificação da aprendizagem

3. Por que a classe `CashRegister` do Capítulo 4 não é coesa?
4. Por que a classe `Coin` não depende da classe `CashRegister`?
5. Por que o acoplamento deve ser minimizado entre as classes?

Dica de qualidade 8.1

Consistência

Nesta seção você aprendeu dois critérios para analisar a qualidade da interface pública de uma classe. Você deve maximizar a coesão e eliminar o acoplamento desnecessário. Há um outro critério que gostaríamos que você prestasse atenção – *consistência*. Quando você tem vários métodos, siga um esquema consistente para os nomes e parâmetros desses métodos. Isso é simplesmente um sinal de bom estilo.

Infelizmente, você pode localizar várias inconsistências na biblioteca padrão. Eis um exemplo. Para mostrar uma caixa de diálogo de entrada, você chama

```
JOptionPane.showInputDialog(promptString)
```

Para mostrar uma caixa de diálogo de mensagem, você chama

```
JOptionPane.showMessageDialog(null, messageString)
```

O que é o parâmetro `null`? Sabemos que o método `showMessageDialog` precisa de um parâmetro para especificar a janela pai ou `null` se nenhuma janela pai for requerida. Mas o método `showInputDialog` não requer nenhuma janela pai. Por que essa inconsistência? Não há razão alguma. Seria mais fácil fornecer um método `showMessageDialog` que espelhasse exatamente o método `showInputDialog`.

Inconsistências como essas não são uma falha fatal, mas são uma dor de cabeça, especialmente porque podem ser facilmente evitadas.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Dica do Professor

Para começarmos um projeto, o primeiro passo é identificar quais objetos devem ser criados para a aplicação. As classes servem para classificar os objetos, como o próprio nome diz.

Existem diversos recursos que precisam ser utilizados dentro de um programa, como os métodos, que definem quais as ações serão executadas. Da mesma forma, as boas práticas de nomeação de classes devem ser adotadas para padronizar a estrutura de um programa.

Nesta Dica do Professor, você vai ver aspectos importantes da POO, selecionando práticas e requisitos que devem ser analisados na fase de planejamento de um programa.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **É importante que o desenvolvedor compreenda a importância de manter as tarefas a serem executadas na construção de um código de maneira organizada. Assim como as boas práticas, os processos também devem ser conduzidos na ordem correta, considerando também a importância de cada um deles.**

Em orientação a objetos, como devemos começar as atividades de programação?

A) Identificando os atributos.

B) Identificando os métodos.

C) Identificando os requisitos funcionais da aplicação.

D) Identificando objetos e as classes às quais eles pertencem.

E) Identificando nomes de classes.

2) **Ao criar um aplicativo, podem ser criadas diferentes classes, o que também traz maior coesão ao programa. Se uma classe com o nome "Cadastro" for criada, ainda não estará claro qual será sua responsabilidade, mas, se uma segunda classe, denominada "Cliente", for criada, é evidente que seu objetivo será tratar da criação do objeto "Cliente".**

Considerando as informações sobre as classes, o que é correto afirmar?

A) Nomearmos uma classe utilizando um verbo que define o objetivo dessa classe.

B) Uma classe deve ser criada para representar vários conceitos do domínio do problema.

C) Se você não pode afirmar, a partir do nome da classe, o que um objeto da classe supostamente deve fazer, provavelmente você não está no caminho certo.

D) Uma categoria útil de classes pode ser descrita como atores. Essas classes servem para iniciar um programa.

E) Uma prática comum é nomear métodos com algum substantivo.

3)

No contexto de desenvolvimento de *software*, em diversas situações, os desenvolvedores passam por problemas ou encontram *bugs* que precisam ser corrigidos, por influenciarem negativamente a funcionalidade de um aplicativo. Para evitar tais defeitos, alguns princípios podem ser aplicados, em conjunto com as boas práticas de programação.

Referente a coesão e acoplamento, o que se pode afirmar?

A) Uma classe coesa representa uma solução bem estruturada no que se refere à criação do objeto.

B) A interface pública de uma classe é coesa se abrange todos os requisitos funcionais do sistema.

C) Quando a interface pública de uma classe referencia vários conceitos, é um bom sinal de que pode ser hora de utilizar classes separadas.

D) Acoplamento refere-se à dependência que as classes têm em relação aos seus métodos.

E) Se muitas classes de um programa dependerem umas das outras, dizemos que o acoplamento entre as classes é baixo.

4) **É uma boa prática evitar conflitos gerados pela nomeação incorreta de classes. Existem recursos que facilitam a escrita do código e auxiliam no processo de manter os objetos e todos os outros componentes atualizados. Os pacotes facilitam a busca ou pesquisa de classes, interfaces e enumerações.**

Ainda sobre os pacotes, o que é correto afirmar?

A) São uma forma de organizar os métodos.

B) São um modificador de acesso.

C) Servem para iniciar programas.

D) Criamos objetos a partir das definições de um pacote.

E) São um conjunto de classes relacionadas.

5) **Analise o código abaixo:**

```
/*
package media;
```

```
public class calcularMedia {

private double nota1;
private double nota2;
private double media;
private int matricula;
private String nome;


public void calcularMedia(double nota1, double nota2){
this.nota1 = nota1;
this.nota2 = nota2;
media = (nota1 + nota2)/2;

}

public void cadastrarAluno(int cod, String matricula){
this.cod=cod;
this.matricula = matricula;
}

}
*/
```

O que é correto afirmar?

A) A classe “calcularMedia” segue a regra geral para nomes de classes.

B) O método “calcularMedia” não irá executar a expressão aritmética.

C) Essa classe não está dentro de nenhum pacote.

D) Essa classe não apresenta coesão.

E) A classe está totalmente escrita de forma correta.

Na prática

A POO é um modelo de programação de *software* em que os objetos manipulados são abstrações de objetos do mundo real. Os objetos são as classes abstraídas do mundo real. Desse modo, um projeto que utiliza o modelo de orientação a objetos é estruturado em pacotes que armazenam classes semelhantes, exigindo que você conheça como é a relação entre essas classes.

Nesta Na Prática, você irá aprender como os pacotes funcionam.

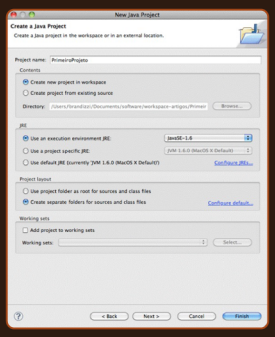
PACOTES NA PROGRAMAÇÃO ORIENTADA A OBJETOS

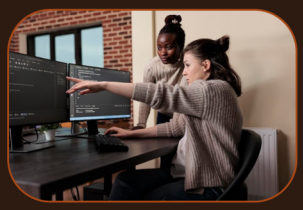
Programas de estruturas complexas são difíceis de administrar e exigem boa organização em relação principalmente à separação das unidades, estejam elas isoladas ou não. Em Java, utilizar objetos pode facilitar o processo. Quando os objetos são compostos por atributos e métodos, precisam estar organizados em pacotes.

Agora você irá aprender a trabalhar com pacotes na programação orientada a objetos.

TRABALHANDO COM PACOTES

Martha está criando um projeto em Java. Ao criar uma classe, no campo **"Name"**, inseriu o nome **"PrimeiraClasse"**. Nesta etapa, visualizou um alerta recomendando a declaração de um pacote (*package*) e deu o nome de **"artigoinicial.meupacote"** no campo rotulado Package.





Maria, que é estagiária, não entendeu por que existe a necessidade de criar um pacote vinculado a uma classe, então Martha explicou **que um pacote pode conter classes e outros pacotes e permite o agrupamento de pastas e diretórios do código-fonte das classes**. João explicou que o pacote criado é composto por dois pacotes – o primeiro deles é denominado **"artigoinicial"**, e o segundo, **"meu pacote"**.

Agora, Maria e Martha podem utilizar pacotes para evitar conflitos, já que é possível utilizar classes produzidas por outras pessoas e organizações em seus projetos. Alguns nomes comuns utilizados pelos pacotes são **File, User e List**. Em Java, toda classe é compilada em um único arquivo, porém, se Maria adicionar sua classe File no pacote br.com.minhaempresa.nomedoprojeto, e Martha adicionar sua classe no pacote **br.org.suaong.projeto**, é possível utilizar ambas as classes em um mesmo programa.

Quando uma classe é criada, o seguinte código poderá ser visto no painel do lado direito, no editor de código:

```
public class PrimeiraClasse {}public
class PrimeiraClasse {}
```

O código define:

- que a classe está vazia; e
- que o pacote está sendo declarado.
- A palavra reservada *public* define que a classe é publica.
- A palavra *class* define o nome da classe.
- As chaves {} definem o corpo da classe.

Como podemos ver, os pacotes são uma boa prática na criação de classes na programação orientada a objetos.

Em desenvolvimento de *software*, é importante saber lidar com as funções e os métodos de uma linguagem, mas também com a organização dos arquivos. Os pacotes permitem separar os códigos em categorias, contribuindo para um projeto ordenado e facilitando a colaboração entre as pessoas que fazem parte de sua construção.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Como criar uma classe com atributos

Veja este vídeo, que ensina a criar uma classe simples utilizando o eclipse, para que você possa praticar. O vídeo aborda dicas importantes para criação de um projeto.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Java: declaração e utilização de classes

O texto aborda a sintaxe para declaração de uma classe. São trazidos exemplos de como utilizar uma classe em Java. Você também verá exemplos de instanciação de uma classe.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

AulaCast Orientação a objetos #6 – Classes e objetos em Java #2

O podcast aborda características que diferenciam classes e objetos, apresentando códigos que aplicam a teoria à prática. Alguns conceitos de modelagem de *software* e abstração também são apresentados.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.