FAESA
CENTRO UNIVERSITÁRIO

Classes aninhadas

Apresentação

Em programação, o termo “aninhada” trata de uma sub-rotina encapsulada em outra. A sub-rotina encapsulada só pode ser chamada pela sub-rotina que a encapsula. Nesse contexto, classes aninhadas (*nested classes*) em Java são classes declaradas dentro de outras classes. Uma classe pode ter várias classes aninhadas, sendo que a própria classe aninhada pode ter outras classes aninhadas a si mesma.

Em orientação a objetos, uma das grandes vantagens de usar classes aninhadas é o ocultamento de informação. Se a classe em questão será utilizada somente por uma classe, não há motivo para que ela fique visível a outras classes. Se uma nova classe é necessária para apenas uma classe, não existe razão para defini-la de forma que seja vista por outras. Nesse caso, a nova classe pode estar aninhada dentro da classe que a usa.

Nesta Unidade de Aprendizagem, você vai conhecer um pouco mais classes aninhadas e internas, seu conceito, características e aplicação.
Bons estudos.

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

- Definir classes aninhadas.
- Identificar a diferença entre classes aninhadas e internas.
- Construir uma aplicação usando classes aninhadas.

Desafio

O uso de classes aninhadas em um código-fonte permite desenvolver uma lógica de grupo das classes que estão sendo utilizadas em um único local. Sua aplicação é indicada para criar uma classe superprotegida, encapsulada em outra classe, para situações em que a classe aninhada não precise ser visível ou reutilizável pelas demais classes do código. Seu conceito está totalmente de acordo com os princípios da programação orientada a objetos, que prevê o encapsulamento e o polimorfismo de classes.

Acompanhe a seguinte situação:

Você trabalha como analista/programador em uma fábrica de *software*. Foi solicitada a criação de uma calculadora com as quatro operações básicas da matemática (soma, subtração, multiplicação e divisão).

Essa calculadora deverá **receber apenas dois números de cada vez e a operação desejada**. Para construir, você deverá fazer o uso de classes aninhadas não estáticas, e cada classe corresponde a uma operação.

A partir do exposto, crie uma classe-controle para instanciar as classes das operações. Para executar a tarefa, utilize uma linguagem de programação orientada a objeto. Logo após a criação, exporte o projeto em um arquivo zipado.

Infográfico

As classes aninhadas têm várias vantagens, entre elas o agrupamento de classes que serão úteis em apenas um local do código. Outra vantagem é o aumento do encapsulamento, que, de acordo com a classificação da classe, permite que a classe aninhada acesse os membros de sua classe envolvente (externa) mesmo que eles estejam privados a outras classes. Além disso, as classes aninhadas deixam o código mais legível e de manutenção mais fácil.

Neste Infográfico, você vai ver conceitos importantes para compreender o funcionamento das classes aninhadas e exemplos relacionados a cada uma delas.

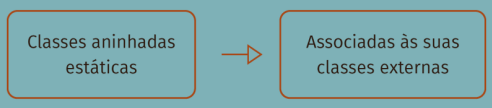
TIPOS DE CLASSES ANINHADAS

As classes aninhadas podem ser divididas em dois tipos principais: estática e não estática (interna comum). As classes aninhadas estáticas não têm acesso aos membros privados da classe externa, apenas dos membros estáticos. Já as classes aninhadas não estáticas têm acesso aos membros privados da classe externa. As classes internas comuns podem ser instanciadas somente se estiverem dentro de um objeto da classe externa.

Veja, a seguir, exemplos da aplicação de classes aninhadas em Java.

CLASSES EXTERNAS OU ENVOLVENTES

- Classes que contêm ou encapsulam as classes internas ou aninhadas.
- Se forem excluídas, suas classes aninhadas também serão.
- Suas classes aninhadas também são seus membros e, por isso, podem ser classificadas como privadas (*private*), públicas (*public*) ou protegidas (*protected*).



Classes aninhadas estáticas

- Não têm acesso aos membros privados não estáticos da classe externa.
- Não podem se referir diretamente a variáveis de instância ou métodos definidos em sua classe externa. Precisam chamá-los com uma referência de objeto.
- Têm a mesma visibilidade dos membros da classe externa que uma classe não encapsulada.

Exemplo de classe aninhada estática:

```
public class Exe01 {
      static int CountStatic = 0;
      int CountNonStatic = 0;

      public static class Inner {
         public void doInnner() {
            System.out.println
            ( CountStatic );
         }
      }
   }
```

O exemplo demonstra de forma simples como utilizar uma classe aninhada estática, chamada “Inner”. Por ser estática, ela não pode chamar diretamente a variável “CountNonStatic”, pois geraria um erro de compilação. Por esse motivo, foi criada uma instância de “Exe01”.

Classes internas

- Classes aninhadas não estáticas, associadas como instâncias de uma classe.
- Têm acesso aos membros privados não estáticos da classe externa.
- Podem se referir diretamente a variáveis de instância ou métodos definidos em sua classe externa. Precisam chamá-los com uma referência de objeto.
- Têm a mesma visibilidade dos membros da classe externa que a própria classe externa, pois também são instanciadas.

Exemplo de classe interna:

```
public class Outer {
   public static void main
   (String[] args) {
      Outer o = new Outer();
      Inner i = o.new Inner();
}

   class Inner {
   }
}
```

O exemplo demonstra a criação do objeto “o” do tipo “Outer” e, em sequência, a criação do objeto “i” do tipo “Inner”, utilizando o objeto “o”. Dessa forma, a classe interna e a externa foram instanciadas.

Como você pode perceber nos exemplos, pequenas diferenças no código-fonte de criação das classes definem se elas são externas, estáticas ou internas. Cada uma delas tem uma função específica para a implementação de um código organizado e de fácil manutenção.

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

Conteúdo do Livro

Classes e interfaces podem ser declaradas dentro de outras classes e interfaces, seja como membros, seja dentro de blocos de código. Essas classes aninhadas e interfaces aninhadas podem ter várias formas diferentes, cada qual com suas propriedades.

Acompanhe um trecho do livro **A linguagem de programação Java**, base teórica para esta Unidade de Aprendizagem. Inicie o estudo pelo tópico Classes e interfaces aninhadas e finalize ao final de Acessando objetos envolventes.

Boa leitura.

Ken Arnold ◆ James Gosling ◆ David Holmes

A Linguagem de Programação Java™

Quarta Edição

Sun microsystems
Bookman®
JAVA™

CAPÍTULO 5

Classes e Interfaces Aninhadas

Cada espaço de produto interno de dimensão finita e não zero possui uma base ortonormal.
Isto faz sentido quando você não pensa sobre isto.
– Professor de Matemática, U.C. Berkeley

Classes e interfaces podem ser declaradas dentro de outras classes e interfaces, seja como membros ou dentro de blocos de código. Estas *classes aninhadas* e *interfaces aninhadas* podem ter várias formas diferentes, cada qual com suas próprias propriedades.

A habilidade de definir tipos aninhados serve a dois objetivos principais. Primeiro, classes aninhadas e interfaces aninhadas permitem que tipos sejam estruturados e acessados em grupos logicamente relacionados. Segundo, e mais importante, classes aninhadas podem ser usadas para conectar, de forma simples e eficiente, objetos logicamente relacionados. Esta última capacidade é usada extensivamente por *frameworks* orientados a eventos, tal como o usado em AWT (ver "`java.awt` – o Abstract Window Toolkit" nas páginas 636-637) e a arquitetura de componentes JavaBeans™ (ver "`java.beans` – Componentes" nas páginas 640-641).

Um tipo aninhado é considerado uma parte de seu tipo envolvente e os dois compartilham um relacionamento de confiança no qual cada um pode acessar os membros do outro. Diferenças entre tipos aninhados dependem de o tipo aninhado ser uma classe ou uma interface, e do tipo envolvente ser uma classe ou uma interface. Tipos aninhados podem ser `static` ou não: o primeiro permite estruturação simples de tipos; o último define um relacionamento especial entre um objeto aninhado e um objeto da classe que o contém. Tipos aninhados estáticos são mais básicos e assim vamos examiná-los primeiro.

5.1 Tipos Aninhados Estáticos

Uma classe ou interface aninhada que é declarada como um membro `static` de sua classe ou interface envolvente atua justamente como qualquer classe ou interface não aninhada, ou de alto nível, exceto que seu nome e escopo são definidos pelo seu *tipo envolvente*. O nome de um tipo aninhado é expresso como `NomeExterno.NomeAninhado`. O tipo aninhado é acessível somente se o tipo envolvente é acessível.

Tipos aninhados estáticos servem como um mecanismo de estruturação e escopo para tipos logicamente relacionados. Entretanto, tipos aninhados estáticos são membros de seu tipo envolvente e, como tal, podem acessar todos os outros membros de seu tipo envol-

vente, incluindo os privados – através de uma referência de objeto apropriada, naturalmente. Isto fornece ao tipo aninhado um relacionamento especial, privilegiado, com seu tipo envolvente.
Porque tipos aninhados estáticos são membros de seu tipo envolvente, as mesmas regras de acessibilidade de outros membros se aplicam a eles. Para classes, isto significa que uma classe ou interface aninhada estática pode ter acesso privado, de pacote, protegido ou público, enquanto que para interfaces, tipos aninhados são implicitamente públicos.

5.1.1 Classes Aninhadas Estáticas

A classe aninhada estática é a forma mais simples de classe aninhada. Você declara uma precedendo a declaração de classe com o modificador `static`. Quando aninhada em uma interface, uma declaração de classe é sempre `static` e o modificador é, por convenção, omitido. Uma classe aninhada estática atua como qualquer outra classe de alto nível. Ela pode estender qualquer outra classe (incluindo a classe da qual é membro),[1] implementar qualquer interface, e ela mesma pode ser usada para extensão posterior por qualquer classe para a qual seja acessível. Ela pode ser declarada `final` ou `abstract`, assim como pode uma classe de alto nível, e pode ter anotações aplicadas a ela.
Classes aninhadas estáticas servem como um mecanismo para definir tipos logicamente relacionados dentro de um contexto no qual este tipo faz sentido. Por exemplo, na página 80 mostramos uma classe `Permissions` que produz informações sobre um objeto `BankAccount`. Visto que a classe `Permissions` é relacionada ao contrato da classe `BankAccount` – ela serve para um objeto `BankAccount` comunicar um conjunto de permissões – ela é uma boa candidata a ser uma classe aninhada:

```
public class BankAccount {
    private long number;    // número da conta
    private long balance;   // saldo atual (em centavos)

       public static class Permissions {
       public boolean canDeposit,
                      canWithdraw,
                      canClose;
    }
    //...
}
```

A classe `Permissions` é definida dentro da classe `BankAccount`, o que a torna um membro desta classe. Quando `permissionsFor` retorna um objeto `Permissions`, ele pode se referir à classe simplesmente como `Permissions`, da mesma maneira que ele pode se referir a `balance` sem qualificação: `Permissions` é um membro da classe. O nome completo da classe é `BankAccount.Permissions`. Este nome completo claramente indica que a classe existe como parte da classe `BankAccount`, e não como um tipo isolado. Código fora da classe `BankAccount` deve usar o nome completo, como por exemplo:

```
BankAccount.Permissions perm = acct.permissionsFor(owner);
```

[1] Isso é uma forma de estruturação muito comum.

Se `BankAccount` estivesse em um pacote denominado `bank`, o nome completo da classe seria `bank.BankAccount.Permissions` (pacotes são discutidos no Capítulo 18). Em seu próprio código, você pode importar a classe `BankAccount.Permissions` e então usar o nome simples `Permissions`, mas você poderia perder a importante informação sobre a natureza subsidiária da classe.

Classes aninhadas estáticas são membros de seu tipo envolvente. Classes aninhadas estáticas envoltas em uma interface são implicitamente públicas; se envoltas por uma classe, você pode declará-las para serem acessíveis da maneira que você quiser. Você pode, por exemplo, declarar uma classe que seja um detalhe de implementação como `private`. Declaramos `Permissions` como sendo `public` porque programadores que usarem `BankAccount` necessitam usar a classe.

Visto que `Permissions` é um membro de `BankAccount`, a classe `Permissions` pode acessar todos os outros membros de `BankAccount`, incluindo todos os membros herdados. Por exemplo, se `Permissions` declarou um método que usa um objeto `BankAccount` como argumento, este método estaria apto a acessar diretamente os campos `number` e `balance` daquela conta. Neste sentido, a classe aninhada é vista como parte da implementação da classe envolvente e por isso é completamente confiável.

Não existe restrição sobre como uma classe aninhada estática pode ser estendida – ela pode ser estendida por qualquer classe à qual ela seja acessível. Naturalmente, a classe estendida não herda o acesso privilegiado que a classe possui na classe envolvente.

Classes enum aninhadas são sempre estáticas, embora por convenção o modificador `static` seja omitido na declaração `enum`. Classes enum são descritas no Capítulo 6.

5.1.2 Interfaces Aninhadas

Interfaces aninhadas são sempre estáticas, e novamente, por convenção, o modificador `static` é omitido da declaração da interface. Elas servem simplesmente como um mecanismo de estruturação para tipos relacionados. Quando examinarmos classes aninhadas não estáticas você verá que elas são inerentemente preocupadas com questões de implementação. Uma vez que interfaces não ditam a implementação, elas não podem ser não estáticas.

Exercício 5.1: Considere a classe `Attr` e a interface `Attributed` do Capítulo 4. Deveria uma destas ser um tipo aninhado da outra? Se sim, qual caminho faria mais sentido?

5.2 Classes Internas

Classes aninhadas não estáticas são chamadas *classes internas*. Membros não estáticos de classes são associados com instâncias de uma classe – campos não estáticos são variáveis de instância e métodos não estáticos operam sobre uma instância. De modo similar, uma classe interna é também (geralmente) associada com uma instância de uma classe, ou mais especificamente, uma instância de uma classe interna é associada com uma instância de sua classe envolvente – a *instância envolvente* ou *objeto envolvente*.

Com freqüência você precisa associar fortemente um objeto da classe aninhada a um objeto particular da classe envolvente. Considere, por exemplo, um método para a classe `BankAccount` que permite que você veja a última ação executada na conta, tal como um depósito ou uma retirada:

```
public class BankAccount {
   private long number;    // número da conta
   private long balance;   // saldo atual (em centavos)
   private Action lastAct; // última ação executada

   public class Action {
      private String act;
      private long amount;
      Action(String act, long amount) {
         this.act = act;
         this.amount = amount;
      }
      public String toString() {
         // identifica sua conta envolvente
         return number + ": " + act + " " + amount;
      }
   }

   public void deposit(long amount) {
      balance += amount;
      lastAct = new Action("deposita", amount);
   }

   public void withdraw(long amount) {
      balance -= amount;
      lastAct = new Action("retira", amount);
   }
   //...
}
```

A classe `Action` registra uma única ação na conta. Ela não é declarada `static`, e isto significa que seus objetos existem como relacionados a um objeto da classe envolvente. O relacionamento entre um objeto `static` e seu objeto `BankAccount` é estabelecido quando um objeto `Action` é criado, como mostrado nos métodos `deposit` e `withdraw`. Quando um objeto de uma classe interna é criado, ele deve ser associado com um objeto de sua classe envolvente. Geralmente, objetos de classes internas são criados dentro de métodos de instância da classe envolvente, como em `deposit` e `withdraw`. Quando isto ocorre o objeto atual `this` é associado com o objeto interno, por *default*. O código de criação em `deposit` é o mesmo que o mais explícito

```
lastAct = this.new Action("deposita", amount);
```

Qualquer objeto `BankAccount` pode ser substituído por `this`. Por exemplo, suponha que adicionemos uma operação de transferência que retira uma quantia específica de uma conta e coloca na conta atual – tal ação necessita atualizar o campo `lastAct` de ambos os objetos de conta:

```
public void transfer(BankAccount other, long amount) {
   other.withdraw(amount);
   deposit(amount);
   lastAct = this.new Action("transfere", amount);
   other.lastAct = other.new Action("transfere", amount);
}
```

Neste caso associamos o segundo objeto `Action` ao outro objeto `BankAccount` e o armazenamos como a última ação da outra conta. Cada objeto `BankAccount` deve somente se referir a objetos `Action` para os quais aquele objeto `BankAccount` é a instância envolvente. Não faria sentido acima, por exemplo, armazenar o mesmo objeto `Action` tanto no campo atual `lastAct` quanto em `other.lastAct`.

Uma declaração de classe interna é justamente como uma declaração de uma classe de alto nível, exceto por uma restrição – classe internas não podem ter membros estáticos (incluindo tipos aninhados estáticos), exceto para campos `final static` que são inicializados com constantes ou expressões construídas a partir de constantes. A razão de permitir que constantes sejam declaradas em uma classe interna é a mesma de permiti-las em interfaces – pode ser conveniente definir constantes dentro do tipo que as utiliza.

Assim como classes de alto nível, classes internas podem estender qualquer outra classe – incluindo sua classe envolvente[2] – implementar qualquer interface e ser estendida por qualquer outra classe. Uma classe interna pode ser declarada `final` ou `abstract`, e podemos ter anotações aplicadas a ela.

Exercício 5.2: Crie uma versão de `BankAccount` que registre as últimas dez ações realizadas na conta. Adicione um método `history` que retorne um objeto `History` que irá retornar objetos `Action`, um de cada vez, através de um método `next`, retornando `null` ao final da lista. Deve `History` ser uma classe aninhada? Se sim, deve ser estática ou não?

5.2.1 Acessando Objetos Envolventes

O método `toString` de `Action` usa diretamente o campo `number` de seu objeto envolvente `BankAccount`. Uma classe aninhada pode acessar todos os membros de sua classe envolvente – incluindo campos e métodos privados – sem qualificação porque ela é parte da implementação da classe envolvente. Uma classe interna pode simplesmente usar os nomes dos membros de seu objeto para utilizá-los. Diz-se que os nomes na classe envolvente estão no escopo. A classe envolvente também pode acessar os membros privados da classe interna, mas somente através de uma referência explícita a um objeto da classe interna tal como `lastAct`. Enquanto um objeto da classe interna é sempre associado com um objeto da classe envolvente, o inverso não é verdadeiro. Um objeto da classe envolvente não necessita ter quaisquer objetos da classe interna associados a ele, ou ele pode ter diversos.

Quando `deposit` cria um objeto `Action`, uma referência ao objeto envolvente `BankAccount` é automaticamente armazenada no novo objeto `Action`. Usando esta referência salva, o objeto `Action` pode sempre se referir ao campo `number` do objeto envolvente

[2] É difícil pensar em uma razão pela qual você poderia querer fazer isto, e fácil ter uma dor de cabeça pensando sobre o que isto significa.

`BankAccount` pelo simples nome `number`, como mostrado em `toString`. O nome da referência ao objeto envolvente é precedido pelo nome da classe envolvente – uma forma conhecida como `this` *qualificada*. Por exemplo, `toString` poderia se referir ao campo `number` do objeto envolvente `BankAccount` explicitamente:

```
return BankAccount.this.number + ": " + act + " " + amount;
```

A referência `this` qualificada reforça a idéia que o objeto envolvente e o objeto interno são fortemente vinculados como parte da mesma implementação da classe envolvente. Isto é posteriormente reforçado pela referência *`super` qualificada*, a qual permite acessar os membros da instância envolvente da superclasse que tenham sido ocultos ou sobrescritos pela classe envolvente. Por exemplo, dada uma classe `T` que estende `S`, dentro de `T` podemos invocar a implementação de um método `m` da superclasse, usando `super.m()` em uma expressão. De modo similar, em uma classe interna de T podemos invocar a mesma implementação de `m` usando `T.super.m()` em uma expressão – uma referência `super` qualificada e de modo similar para campos de `S` ocultos por campos de `T`.

Uma classe aninhada pode ter suas próprias classes e interfaces aninhadas. Referências a objetos envolventes podem ser obtidas para qualquer nível de aninhamento da mesma maneira: o nome da classe e `this`. Se a classe `X` envolve a classe `Y` que envolve a classe `Z`, código em `Z` pode explicitamente acessar campos de `X` usando `X.this`.

A linguagem não evita que você faça classes profundamente aninhadas, mas o bom gosto deveria. Uma classe duplamente aninhada tal como `Z` possui três escopos de nomes: ela mesma, sua classe imediatamente envolvente `Y` e a classe mais externa `X`. Alguém que leia o código de `Z` deve entender cada classe extensivamente para saber a qual contexto um identificador é vinculado e qual objeto envolvente foi vinculado a qual objeto aninhado. Recomendamos aninhar somente um nível na maior parte das circunstâncias. Aninhar mais do que dois níveis convida a um desastre de legibilidade e provavelmente nunca deve ser tentado.

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

As classes aninhadas têm a característica de serem declaradas dentro de outras classes. A aplicação desse conceito aumenta a produtividade do projeto de *software*, melhora a legibilidade do código-fonte e provê maior facilidade de manutenção, se utilizado de maneira correta. As classes aninhadas são divididas basicamente em estáticas e internas. As estáticas somente conseguem acessar diretamente os membros estáticos da classe externa, e não os membros categorizados como privados. Já as classes externas podem acessar esses dois tipos de membros da classe externa.

Nesta Dica do Professor, veja um exemplo de criação de uma classe externa com duas classes aninhadas instanciadas, uma estática e outra interna.

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

Exercícios

1) **O uso de classes aninhadas contribui para o desenvolvimento de *software* em vários aspectos. Sobre o conceito de classes aninhadas, veja as afirmações a seguir:**

I. Classes aninhadas são declaradas dentro de um mesmo projeto.
II. Classes aninhadas são declaradas dentro de outras classes.
III. Classes aninhadas são declaradas com a palavra-chave *abstract*.
IV. Classes aninhadas dependem da existência de outra classe para serem declaradas.

Está correto o que se afirma em:

A) II e IV.

B) I e IV.

C) I, III e IV.

D) III e IV.

E) II, III e IV.

2) **As classes aninhadas são úteis em diferentes tipos de sistemas e aplicações e têm características específicas. Sobre o conceito e as características de classes externas, veja as afirmações a seguir:**

I. Classes externas são classes declaradas com o modificador de acesso *public*.
II. Classes aninhadas classificadas como *protected* são protegidas.
III. Classes externas são classes que contêm classes aninhadas.
IV. Classes externas são todas aquelas que têm membros como atributos e métodos.

Está correto o que se afirma em:

A) II e IV.

B) I, II e IV.

C) II e III.

D) III e IV.

E) I, III e IV.

3) **As classes aninhadas podem ser divididas em dois tipos principais que apresentam características diferentes. Nesse sentido, observe os tipos de classes aninhadas a seguir e os relacione com suas respectivas características:**

I. Classes aninhadas internas
II. Classes aninhadas estáticas

( ) Têm acesso aos membros privados não estáticos da classe externa.
( ) Chamam as variáveis de instância de sua classe externa com uma referência de objeto.
( ) Membros estáticos podem ser declarados dentro delas.
( ) Referem-se diretamente às variáveis de instância definidas em sua classe externa.

Assinale a alternativa em que é apresentada a ordem correta.

A) II, I, II, I.

B) II, I, I, II.

C) I, II, I, II.

D) I, II, II, I.

E) II, II, I, I.

4) **A aplicação de classes aninhadas em um código-fonte depende da utilização de uma sintaxe de código específica, a fim de garantir sua funcionalidade. Nesse sentido, observe o código-fonte a seguir:**

```
1 public class BankAccount {
2    private long number;
3   private long balance;
4
5   public static class Permissions {
6    public boolean canDeposit, canWithdraw, canClose;
7
8   }
9   //..
10 }
```

Agora, veja as afirmações a seguir sobre o código apresentado:
I. A classe Permissions é declarada dentro da classe BankAccount.
II. A classe Permissions é um membro da classe BankAccount.
III. Se a classe BankAccount estivesse dentro do pacote banco, o nome completo da classe seria BankAccount.Permissions.
IV. A classe Permissions é proibida de acessar os membros de BankAccount.

Está correto o que se afirma em:

A) I, II e III.

B) II e IV.

C) II, III e IV.

D) II e III.

E) I e II.

5) **Existem algumas maneiras de instanciar uma classe aninhada não estática. Considere uma classe externa chamada A e uma classe aninhada chamada B.**

Qual alternativa tem o código correto para essa situação?

A) A variavel = new A();

B) B variavel = new B();

C) B variavel = new A().new B();

D) A.B variavel = new A().new B();

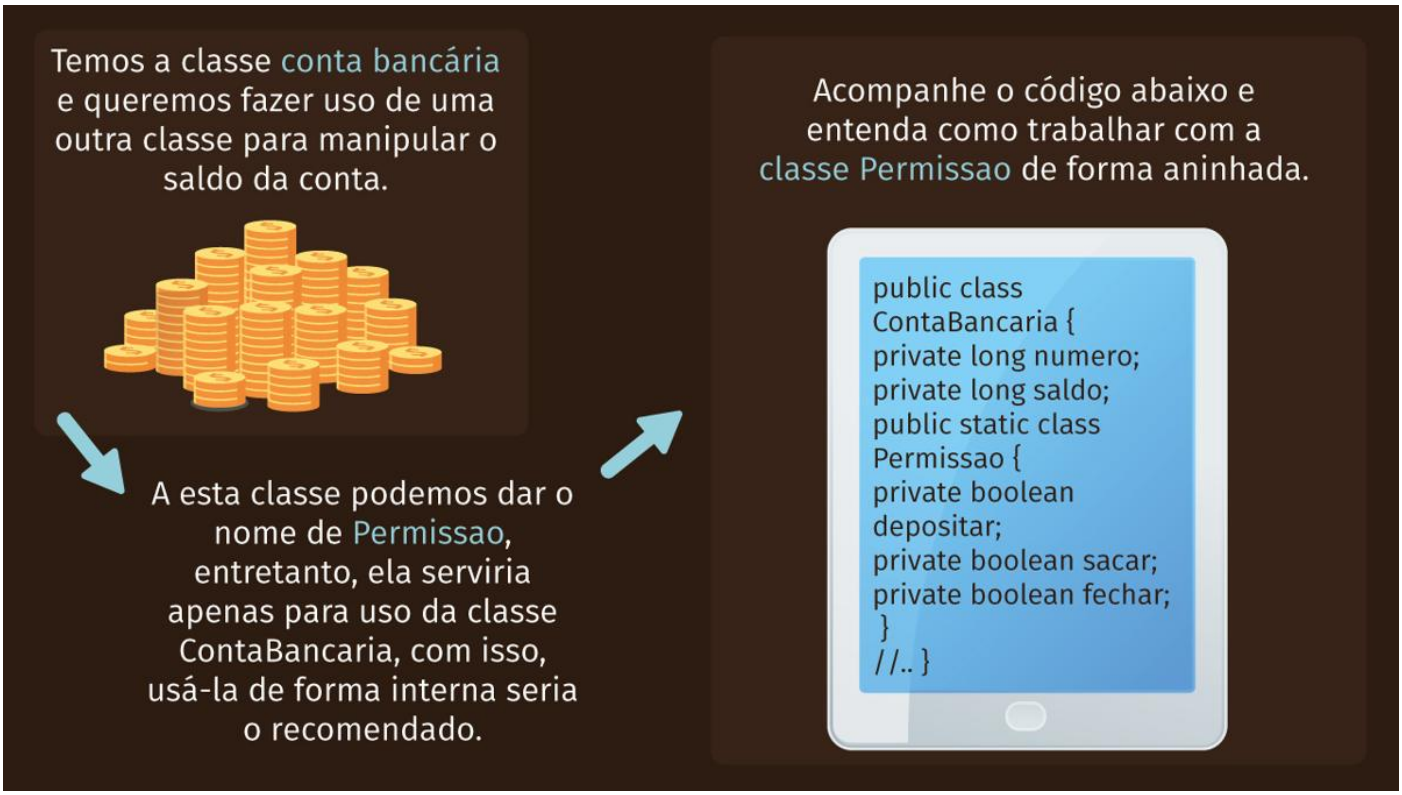
E) A variavel = new B().new A();

Na prática

Classes aninhadas são usadas nos casos em que você tem uma classe X e somente uma classe Y irá utilizá-la; assim, é melhor criar a classe X como interna da classe Y. As classes aninhadas aumentam o encapsulamento, pois podemos declarar seus membros como privados (*private*), públicos (*public*) ou protegidos (*protected*), sendo que eles serão de uso da classe externa.

As classes internas podem ser classificadas como comuns, internas a um método e internas anônimas. As classes internas a um método só podem ser inseridas e instanciadas dentro de um método específico, não sendo possível realizar a instanciação da referida classe fora do método escolhido ou fora da classe externa. Já as classes internas anônimas são a herança de determinada classe a um local exclusivo.

Neste Na Prática, você vai ver um exemplo de código-fonte para uma aplicação de classes internas privadas em uma operação bancária.



Saiba mais

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

Classes internas

Neste vídeo, você vai ver conceitos, vantagens e um exemplo comentado de códigos-fonte com classes internas. Não esqueça de ativar as legendas em português.

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

Introdução: sobrecarga de métodos e tipos genéricos em Java

Veja, neste artigo, como e quando utilizar sobrecarga de métodos e tipos genéricos e quais problemas cada um ajuda a resolver.

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

Classes aninhadas no tratamento de eventos

Para conhecer mais conceitos e exemplos do uso de classes aninhadas em rotinas de tratamento de evento em Java, consulte o conteúdo presente nas páginas 650 a 664 do livro *Android: como programar*.

Conteúdo interativo disponível na plataforma de ensino!