Interfaces

Apresentação

Em programação orientada a objetos, o conceito de interface é utilizado para isolar o mundo externo dos componentes tratados pela linguagem de programação empregada. O principal objetivo é que os detalhes de implementação de um componente de *software* sejam suportados pelo objeto por meio dos métodos.

Um modelo de interface é caracterizado por uma classe que tem métodos não implementados. Entretanto, ao passo que se declara a palavra-chave *class* para classes, declara-se interface para interface. Dessa forma, podemos exemplificar interface como sendo um contrato. E outras classes podem assinar esse contrato, responsabilizando-se pela implementação desses métodos.

Nesta Unidade de Aprendizagem, você vai conhecer um pouco mais sobre interfaces e sua aplicação em códigos construídos utilizando a linguagem Java.

Bons estudos.

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

- Definir interface.
- Identificar herança e implementação de interface.
- Construir uma aplicação usando interfaces.

Desafio

As interfaces podem ser utilizadas até mesmo em tarefas mais simples. Além de definirem métodos a serem usados pelas classes que as implementam, as interfaces podem declarar constantes para serem usadas por essas classes. Na linguagem Java, a interface é utilizada para garantir que determinado grupo de classes tenha propriedades e métodos em comum para definir dado contexto.

Analise o seguinte cenário:

Você trabalha na área de tecnologia da informação (TI) em uma escola. A secretária da instituição solicitou a você a criação de um recurso no sistema que permita inserir a nota obtida pelo aluno em determinada disciplina. A nota deve ser definida em um campo específico em formato decimal, porém retornando ao cliente, após inserção do valor, o resultado obtido.

Você deverá criar uma interface com três constantes:

1) "Parabéns, você atingiu todos os indicadores de avaliação com excelência";

2) "Parabéns, você obteve aproveitamento satisfatório nos indicadores de avaliação";

3) "Você não atingiu o mínimo esperado para aprovação".

Para atender ao pedido, você deverá:

- utilizar Java na construção do código;

- criar uma classe para implementar a interface;

- criar uma variável na classe para receber a opção digitada pelo usuário;

- utilizar o método showInputDialog() da classe JOptionPane para solicitar as opções;

- criar uma estrutura de decisão para verificar a opção digitada e printar na tela o conceito e sua descrição;

- criar um projeto denominado ConceitoAluno; dentro dele, criar um pacote chamado interfaces, e, dentro do pacote, a interface e a classe para implementá-la.

Infográfico

Em programação, linguagens como Java utilizam conceitos importantes para construção de programas. Uma interface é composta por variáveis e métodos abstratos, que obrigam as classes que a implementam a utilizarem esses membros — entretanto, podem fazer uso de forma diferente.

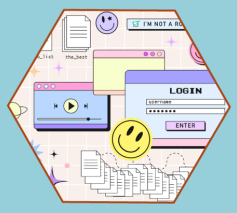
Por sua vez, uma herança é um mecanismo aplicado para estender as classes existentes adicionando novos métodos e campos. Para implementar uma interface, as classes expandem seus próprios tipos como uma estrutura de herança.

Observe o Infográfico a seguir para compreender os conceitos de interface e como ela é aplicada em conjunto com a herança.

O poder da interface

Uma interface representa um contrato estabelecido entre o mundo exterior e uma classe criada pela linguagem de programação, porém o conceito de interface vai muito além disso.

Veja como funciona:



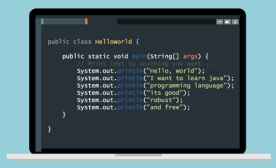
Por que usar interfaces?

- Para definir classes e o comportamento de um objeto.
- Para determinar as funcionalidades que dependem das classes que implementam interfaces e dão origem aos métodos.

Obs.: *Structs* são o meio de implementação das interfaces.

Implementação de *interfaces*

- Uma ou mais **interfaces** podem ser implementadas em uma mesma classe.
- As interfaces devem ser delimitadas por vírgulas.
- Quando uma classe implementa uma interface, deve declarar todos os métodos definidos pela classe para evitar erros de compilação.



Neste código, foi criada uma classe denominada “funcionário”, cujo objetivo é o reajuste salarial com base no salário atual * 1.50.

```
public class Funcionario
    {
        interface IReajuste
        {
            double Reajuste();
        }

        public class ReajusteFuncionario : IReajuste
        {
            //Propriedades públicas da minha classe
            public string nomeFuncionario;
            public double salarioFuncionario;

            //Implementação do método Reajuste, da interface IReajuste
            public double Reajuste()
            {
                salarioFuncionario = salarioFuncionario * 1.50;

                return salarioFuncionario;
            }
        }
    }
```

A interface tem um método único, aplicado à classe, que herda da interface. O contrato define o método Reajuste() e a classe implementa o método construído.

A interface seria implementada da seguinte forma:

```
static void Main(string[] args)
{
            //a classe ReajusteFuncionario é instanciada e
            herda características da interface IReajuste
            Funcionario.ReajusteFuncionario objReajusteFuncionario =
            new Funcionario.ReajusteFuncionario();

            //os valores são atribuídos para o nome e salário
            objReajusteFuncionario.nomeFuncionario = "Felipe Souza";
            objReajusteFuncionario.salarioFuncionario = 2500.00;

            //os dados são escritos na tela
            Console.WriteLine("Antes do Reajuste: ");
            Console.WriteLine("\nNome: " +
            objReajusteFuncionario.nomeFuncionario);
            Console.WriteLine("\nSalário: R$ " +
            objReajusteFuncionario.salarioFuncionario);

            //o método Reajuste é invocado e implementado na classe
            // a interface é criada
            objReajusteFuncionario.Reajuste();

            Console.WriteLine("\nDepois do Reajuste: ");
            Console.WriteLine("\nNome: " +
            objReajusteFuncionario.nomeFuncionario);
            Console.WriteLine("\nSalário: R$ " +
            objReajusteFuncionario.salarioFuncionario);

           //Paro a execução até que alguma tecla seja pressionada
           Console.ReadKey();
```

Conclui-se que uma interface em Java é semelhante a uma classe abstrata, porém composta somente por métodos abstratos, não podendo ser instanciada.

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

Conteúdo do Livro

Interfaces definem tipos em uma forma abstrata, como uma coleção de métodos ou outros que formam o contrato para determinado tipo. Interfaces não contêm implementações e você não pode criar instâncias de uma interface.

Inicie o estudo pelo tópico **Interfaces** e finalize com a leitura do tópico **Estendendo interfaces**, da obra *A linguagem de programação Java*, base teórica desta Unidade de Aprendizagem, e confira mais detalhes sobre o tema.

Boa leitura.

Ken Arnold ◆ James Gosling ◆ David Holmes

A Linguagem de Programação Java™

Quarta Edição

Sun microsystems
Bookman®
JAVA™

Ken Arnold, ex-engenheiro sênior do Sun Microsystems Laboratories, é um grande especialista em projeto e implementação orientada a objetos. Ele foi um dos arquitetos originais da tecnologia Jini™ e engenheiro chefe da tecnologia JavaSpaces™ da Sun.

James Gosling é um Sun Fellow e CTO do Grupo de Desenvolvimento da Plataforma da Sun Microsystems. Projetou a linguagem de programação Java original, implementou seu compilador e a máquina virtual originais, e, mais recentemente, contribuiu para a Real-Time Specification de Java. Um dos mais respeitados programadores da indústria, ele recebeu o prêmio de Excelência em Programação da *Software Development* do ano de 1996.

David Holmes é diretor da DLTech Pty Ltd, localizada em Brisbane, Austrália. Especialista em sincronização e concorrência, participou do grupo de especialistas JSR-166 que desenvolveu os novos utilitários de concorrência. Ele também contribuiu para a atualização da Real-Time Specification de Java e passou os últimos anos trabalhando em uma implementação desta especificação.

A756l Arnold, Ken
A linguagem de programação Java [recurso eletrônico] / Ken Arnold, Jamoes Gosling, David Holmes ; tradução Maria Lúcia Blank Lisbôa. – 4. ed. – Dados eletrônicos. – Porto Alegre : Bookman, 2007.

Editado também como livro impresso em 2007.
ISBN 978-85-60031-61-0

1. Computação – Linguagem de programação. I. Gosling, James. II. Holmes, David. III. Título.

CDU 004.438JAVA

Catalogação na publicação: Juliana Lagôas Coelho – CRB 10/1798

CAPÍTULO 4

Interfaces

"Reger" é quando você desenha "figuras" em nenhum lugar – com sua batuta ou com suas mãos – que são interpretadas como "mensagens instrucionais" por rapazes vestindo gravatas borboleta que desejariam estar pescando.
– Frank Zappa

A unidade fundamental de programação na linguagem de programação Java é a *classe*, mas a unidade fundamental de projeto orientado a objetos é o *tipo*. Enquanto classes definem tipos, é bastante útil e poderoso poder definir um tipo sem definir uma classe. Interfaces definem tipos em uma forma abstrata como uma coleção de métodos ou outros tipos que formam o contrato para aquele tipo. Interfaces não contêm implementações e você não pode criar instâncias de uma interface. Em vez disso, classes podem expandir seus próprios tipos pela *implementação* de uma ou mais interfaces. Uma interface é uma expressão de projeto puro, enquanto que uma classe é uma mistura de projeto e implementação.

Uma classe pode implementar os métodos de uma interface de qualquer maneira que o projetista da classe escolher. Uma interface, portanto possui muitas mais possíveis implementações do que uma classe. Cada classe principal em uma aplicação deve ser uma implementação de alguma interface que captura o contrato daquela classe.

Classes podem implementar mais de uma interface. A linguagem de programação Java permite herança múltipla de interface, mas somente herança simples de implementação – uma classe pode estender somente uma outra classe. Classes podem usar herança de interfaces para expandir seu tipo e então usar, por exemplo, composição para fornecer uma implementação para estas interfaces. Este projeto permite a flexibilidade de tipo da herança múltipla ao mesmo tempo em que evita os perigos da herança múltipla de implementação, ao custo de algum trabalho adicional para o programador.

Em uma dada classe, as classes que são estendidas e as interfaces que são implementadas são coletivamente chamadas de *supertipos*, e, do ponto de vista dos supertipos, uma nova classe é um *subtipo*. A nova classe inclui todos os seus supertipos, de modo que a referência a um objeto do subtipo pode ser usada polimorficamente em qualquer lugar onde uma referência a um objeto de qualquer de seus supertipos (classe ou interface) é requerida. Declarações de interfaces criam nomes de tipos assim como fazem as declarações de classes; você pode usar o nome de uma interface como o nome de tipo de uma variável, e qualquer objeto cuja classe implemente esta interface pode ser atribuído a esta variável.

4.1 Um Exemplo de uma Interface Simples

Muitas interfaces simples definem uma propriedade que é atribuível a diversos objetos de diferentes classes. Estas propriedades são frequentemente definidas em termos de um objeto estar "apto" a fazer algo. Por exemplo, nos pacotes padrões existem diversas interfaces de "habilidade", tais como;

- `Cloneable` – objetos deste tipo suportam clonagem, como você aprendeu com detalhes na página 114.
- `Comparable` – objetos deste tipo possuem um ordenamento que permite que sejam comparados.
- `Runnable` – objetos deste tipo representam uma unidade de trabalho, que muitas vezes podem ser executadas em um fluxo de controle independente (ver Capítulo 14).
- `Serializable` – objetos deste tipo podem ser escritos em um *stream* de bytes de objetos para serem remetidos para uma nova máquina virtual, ou para armazenamento persistente e posterior reconstituição para um objeto vivo (ver "Serialização de Objetos" nas páginas 492-493).

Vamos examinar com mais detalhes a interface `Comparable`. Esta interface pode ser implementada por qualquer classe cujos objetos podem ser comparados uns com os outros de acordo com o "ordenamento natural" da classe. A interface contém um único método:

```
public interface Comparable<T> {
   int compareTo(T obj);
}
```

Uma declaração de interface é similar a uma declaração de classe, exceto que a palavra-chave `interface` é usada em lugar de `class`. Também existem regras especiais que governam os membros de uma interface, como em breve você vai aprender.
O método `compareTo` usa como argumento um único objeto do tipo `T` e o compara ao objeto atual (esperado ser também do tipo `T`), retornando um inteiro negativo, nulo ou positivo, se o objeto atual é menor do que, igual a ou maior do que o argumento, respectivamente.
Considere uma variação da classe `Point` que introduzimos no Capítulo 1. O ordenamento natural para pontos poderia ser a sua distância da origem. Poderíamos então tornar `Comparable` objetos `Point`:

```
class Point implements Comparable<Point> {

   /** Referência da origem que nunca muda */
   private static final Point ORIGIN = new Point();

   private int x, y;

   //... definição de construtores, métodos de configuração e
   de acesso

   public double distance(Point p) {
      int xdiff = x - p.x;
```

```
        int ydiff = y - p.y;
        return Math.sqrt(xdiff * xdiff + ydiff * ydiff);
    }

    public int compareTo(Point p) {
        double pDist = p.distance(ORIGIN);
        double dist = this.distance(ORIGIN);
        if (dist > pDist)
            return 1;
        else if (dist == pDist)
            return 0;
        else
            return -1;
    }
}
```

Primeiro declaramos que `Point` é uma classe `Comparable`. A classe identifica os tipos de interface que ela implementa relacionando-os após a palavra-chave `implements`, antes que o corpo da classe seja definido (e após a cláusula `extends`). Todas estas interfaces são as *superinterfaces* da classe. A classe deve providenciar uma implementação para todos os métodos definidos em suas superinterfaces, ou senão a classe deve ser declarada como `abstract`, e assim exigindo que qualquer subclasse não abstrata os implemente.

O único método que necessitamos implementar para a interface `Comparable` é `compareTo`, e para realmente comparar dois pontos simplesmente comparamos as suas distâncias em relação à origem.

Interfaces introduzem nomes de tipos assim como fazem as classes, de modo que você pode declarar variáveis destes tipos. Por exemplo:

```
Comparable<Point> p1;
```

De fato, muito do poder das interfaces provém da declaração e uso de variáveis somente do tipo da interface, em vez de algum tipo específico de classe. Por exemplo, você pode definir uma rotina `sort` de uso geral, que pode ordenar qualquer array de objetos `Comparable` sem se preocupar com o que a classe desses objetos é na realidade – todos os objetos do array devem, naturalmente, ser do mesmo tipo:[1]

```
class Sorter {
    static Comparable<?>[] sort(Comparable<?>[] list) {
        // detalhes de implementação...
        return list;
    }
}
```

Referências do tipo interface, entretanto, podem ser usadas somente para acessar membros daquela interface. Por exemplo, o seguinte irá produzir um erro de compilação:

```
Comparable<Point> obj = new Point();
double dist = obj.distance(p1); // INVÁLIDO: Comparable não
                                // possui método distance
```

[1] No capítulo 11 você aprende sobre métodos genéricos, e `sort` realmente deveria ser definido como um desses.

Se você quiser tratar `obj` como um objeto `Point` você deve fazer uma coerção explícita para este tipo.
Você pode invocar quaisquer dos métodos de `Object` usando uma referência do tipo de uma interface porque não importa quais interfaces o objeto implemente, ele sempre será um `Object` e assim possui aqueles métodos. De fato, qualquer interface que não estende outra interface explicitamente possui membros que são os métodos públicos de `Object` (a menos que a interface os sobrescreva explicitamente). Portanto, o seguinte é legal:

```
String desc = obj.toString();
```

como uma atribuição de uma referência de interface a uma referência de `Object`.

4.2 Declarações de Interfaces

Uma interface é declarada usando a palavra-chave `interface`, fornecendo um nome à interface e relacionando os membros da interface entre chaves.
Uma interface pode declarar três tipos de membros:

- constantes (campos)
- métodos
- classes e interfaces aninhadas

Todos os membros da interface são implicitamente públicos, mas, por convenção, o modificador `public` é omitido. Ter membros não públicos em uma interface teria pouco sentido; onde isto faz sentido você pode usar a acessibilidade da própria interface para controlar o acesso aos membros da interface.
Vamos postergar a discussão de classes e interfaces aninhadas até o Capítulo 5.

4.2.1 Constantes de Interfaces

Uma interface pode declarar constantes denominadas. Estas constantes são definidas como campos, mas são implicitamente `public`, `static` e `final` – novamente, por convenção, os modificadores são omitidos nas declarações de campos. Estes campos devem ter inicializadores – *brancos finais* não são permitidos. Anotações também podem ser aplicadas aos campos – ver Capítulo 15.
Visto que interfaces não contêm detalhes de implementação, elas não podem definir campos normais – tal definição estaria ditando uma política de implementação para classes que escolhessem implementar a interface. Interfaces podem definir constantes denominadas porque elas são úteis no projeto de tipos. Por exemplo, uma interface que possui diferentes níveis de verbosidade em seu contrato pode definir o seguinte:

```
interface Verbose {
   int SILENCIOSA = 0;
   int CONCISA = 1;
   int NORMAL = 2;
   int PROLIXA = 3;

   void setVerbosity(int level);
   int getVerbosity();
}
```

SILENCIOSA, CONCISA, NORMAL e PROLIXA podem ser passadas ao método setVerbosity, dando nomes a valores constantes que representam significados específicos. Neste caso particular, os níveis de verbosidade podem ser representados mais efetivamente por constantes de um tipo enumerado aninhado dentro da interface Verbose. Tipos enumerados são discutidos no Capítulo 6.
Se você necessita dados compartilhados e modificáveis em sua interface, você pode conseguir isto usando uma constante denominada que se refere a um objeto que contém o dado. Uma classe aninhada é boa para definir tal objeto, de modo que postergaremos um exemplo até o Capítulo 5.

4.2.2 Métodos de Interfaces

Os métodos declarados em uma interface são implicitamente abstratos porque nenhuma implementação é, ou pode ser, dada a eles. Por esta razão, o corpo do método é simplesmente um ponto-e-vírgula após o cabeçalho do método. Por convenção, o modificador abstract é omitido na declaração do método.
Nenhum outro modificado, de método é permitido em uma declaração de método de interface, exceto para anotações – ver Capítulo 15. Eles são implicitamente public e assim não podem ter nenhum outro modificador de acesso. Eles não podem ter modificadores que definem características de implementação – tais como native, synchronized ou strictfp – porque uma interface não dita implementação e eles não podem ser final porque ainda não foram implementados. Naturalmente, a implementação destes métodos dentro de uma classe específica pode ter qualquer um destes modificadores. Métodos de interfaces nunca podem ser métodos static porque métodos static não podem ser abstract.

4.2.3 Modificadores de Interfaces

Uma declaração de interface pode ser precedida de modificadores de interface:

- *anotações* – anotações e tipos de anotação são discutidos no Capítulo 15.
- *pública* – uma interface public é publicamente acessível. Sem este modificador, uma interface somente é acessível dentro de seu próprio pacote.
- *abstratas* – todas as interfaces são implicitamente abstract porque seus métodos são todos abstratos – eles não possuem implementação. Novamente, por convenção, o modificador abstract é sempre omitido.
- *ponto-flutuante estrito* – uma interface declarada strictfp possui toda a aritmética de ponto-flutuante, definida dentro da interface, avaliada estritamente. Isto se aplica a expressões de inicialização para constantes e para todos os tipos aninhados declarados na interface. Em contraste com classes, isto não implica que cada método na interface seja implicitamente strictfp, porque este é um detalhe de implementação. Ver Seção 9.1.3 nas páginas 199-200 para detalhes.

Quando diversos modificadores são aplicados à mesma interface, recomendamos usar a ordem relacionada acima.

4.3 Estendendo Interfaces

Interfaces podem ser estendidas usando a palavra-chave `extends`. Interfaces, diferentemente de classes, podem estender mais de uma interface:

```
public interface SerializableRunnable
                 extends java.io.Serializable, Runnable
{
   //...
}
```

A interface `SerializableRunnable` estende `java.io.Serializable` e `Runnable` ao mesmo tempo, o que significa que todos os métodos e constantes definidos nestas interfaces agora fazem parte do contrato de `SerializableRunnable`, juntamente com todos os novos métodos e constantes que ela define. As interfaces que são estendidas são as *superinterfaces* da nova interface e a nova interface é uma *subinterface* de suas superinterfaces.

Visto que interfaces suportam herança múltipla, o grafo de herança pode conter diversos caminhos para a mesma superinterface. Isto significa que constantes e métodos podem ser acessados por diversos caminhos. Entretanto, visto que interfaces não definem implementação de métodos e não fornecem campos de objetos, não existem dificuldades concernentes a esta forma de herança múltipla.

Dica do Professor

Web Services é um conjunto de métodos específicos que outras aplicações podem invocar. Pode ser utilizado para transferência de dados de protocolos de comunicação entre diferentes plataformas, independentemente da linguagem de programação.

Nesta Dica do Professor, saiba mais sobre como as interfaces podem ser aplicadas aos Web Services. Assim, você poderá entender melhor sobre o seu funcionamento.

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

Exercícios

1) **Todos os componentes têm um papel importante na linguagem de programação orientada a objetos. Porém, um deles é considerado unidade fundamental de uma estrutura de código.**

Considerando o conceito apresentado, assinale a alternativa que contém o recurso que representa tal unidade.

A) Método.

B) Atributos.

C) Modificadores de acesso.

D) Pacote.

E) Classes.

2) **Diferentemente de uma classe, uma interface em Java não suporta qualquer tipo de código, ao passo que uma classe abstrata pode prover um código completo ou ter apenas a declaração de um modelo a ser sobrescrito.**

Assinale a alternativa que apresenta a definição de interface.

A) São formulários que interagem com o usuário.

B) Interface define tipos em forma abstrata.

C) São elementos da classe.

D) Interface serve para organizar classes de uma aplicação.

E) Interfaces são classes que têm apenas métodos e podem ser instanciadas.

3) **Em uma estrutura de herança, uma classe usa membros de outra classe. Interfaces fornecem membros para que outras classes possam fazer uso deles. Ao implementar heranças e interfaces, algumas diferenças precisam ser notadas.**

Sobre a herança, assinale a alternativa correta.

A) Herança é uma estrutura que tem uma superclasse e subclasses que herdam membros dessa superclasse.

B) Em uma estrutura de herança, as subclasses herdam apenas métodos da superclasse.

C) Ao herdar um método da superclasse, ele não será sobrescrito pela superclasse.

D) Uma superclasse não pode ser instanciada.

E) Para criar uma estrutura de herança, o número de subclasses deve ser limitado a dois.

4) **Ao criar uma interface com o nome Contrato e com o método sem retorno chamado entrarComTexto(), é necessário definir alguns componentes.**

A forma correta de criar a interface em Java é:

A) public class Contrato{ void entrarComTexto(String texto); }

B) public abstract class Contrato{ void entrarComTexto(String texto); }

C) public Interface class Contrato{ void entrarComTextoString texto;}

D) public interface Contrato{ void entrarComTexto(String texto); }

E) public interface Contrato{ void entrarComTexto(String texto){ text = texto; System.out.println(texto); } }

5) **Uma classe que implementa uma interface assume a responsabilidade de executar as ações que a interface define. A é a classe e B é a interface.**

Dessa forma, para obter as assinaturas dos métodos da interface em uma classe, analise os códigos a seguir e identifique a forma correta de implementação.

A) public class A extends B{ }

B) public implements A class B { }

C) public class A extends C implements B { }

D) public class A implements B { }

E) public interface B { }

Na prática

Uma interface é um recurso da orientação a objetos muito utilizado em Java para obrigar as classes que a implementam a utilizarem seus métodos. Para compreender a sua aplicação, pode-se fazer uma analogia a um contrato de prestação de serviços. Nesse contrato, existem cláusulas que a contratada deverá seguir obrigatoriamente. As classes em Java seriam a contratada e as interfaces, o contrato.

Ao falar que uma classe está implementando uma interface, imagine como se ela estivesse assinando um contrato, comprometendo-se em usar os métodos da interface.

Neste Na Prática, veja como implementar uma interface com base no conceito de contrato.

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:

Linha de código: utilizando interfaces

Acesse para saber mais a respeito de interfaces em Java.

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

Interface em Java (*implements*): o que é, para que serve e como implementar

Neste *site*, você vai conferir informações a respeito de uma interface de Java: *implements*. Confira.

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

Java interface: aprenda a usar corretamente

Leia o artigo a seguir e conheça mais a respeito do uso de interfaces em Java.

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

Interface em Java POO

Aprenda como aplicar interfaces em Java por meio de exemplos em que classes herdam especificações implementadas por outras classes.

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