PROGRAMAÇÃO ORIENTADA A OBJETOS

CLASES ABSTRATAS DE INTERFACES

AGENDA

1. Classes Abstratas
2. Métodos Abstratos
3. Classes Abstratas como Tipos
4. Interfaces
5. Interfaces como Tipos

CLASSES ABSTRATAS

- Uma **classe abstrata** é uma classe que não é concebida para criar instâncias;
- Seu propósito é servir como uma superclasse para outras classes;
- As classes abstratas podem conter **métodos abstratos**.

CLASSES ABSTRATAS

- Para uma **subclasse** de uma classe abstrata tornar-se **concreta**, ela deve fornecer implementações para todos os métodos abstratos herdados;
- Caso contrário, ela própria é abstrata.
- As classes que não são abstratas são chamadas **classes concretas**.

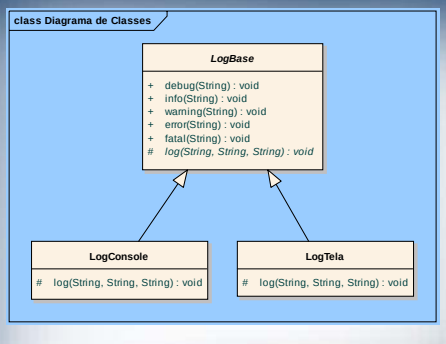
CLASSES ABSTRATAS

- Declarar uma classe abstrata serve a vários propósitos:
  - Nenhuma instância pode ser criada das classes abstratas;
  - Somente classes abstratas podem ter métodos abstratos;
  - As classes abstratas com métodos abstratos forçam as subclasses a sobrescreverem e implementarem os métodos declarados como abstratos.

MÉTODOS ABSTRATOS

- A definição de um **método abstrato** consiste em uma assinatura de método sem um corpo de método;
- Um **método abstrato** é caracterizado por dois detalhes:
  - Ele é prefixado com a palavra-chave ***abstract***;
  - Ele não tem um corpo de método;
  - Seu cabeçalho termina com um ponto-e-vírgula.

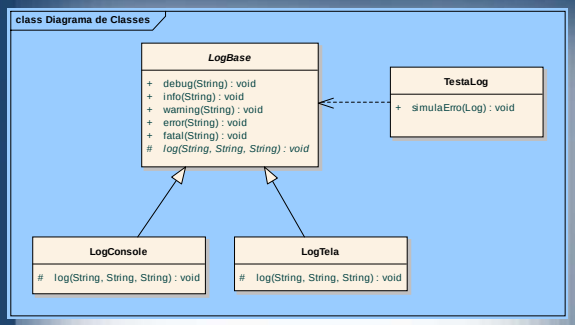
EXEMPLO LOG



CLASSES ABSTRATAS COMO TIPOS

- Atributos e variáveis podem ser declaradas utilizando um tipo abstrato;
- Eles podem referenciar qualquer subtipo da tipo abstrato correspondente.

EXEMPLO LOG



HERANÇA MÚLTIPLA

- Uma situação em que uma classe herda de mais de uma superclasse é chamada de **herança múltipla**;
- A subclasse tem então todos os recursos das suas superclasses e das que são definidas na própria subclasse.

HERANÇA MÚLTIPLA

- Diferentes linguagens orientadas a objetos variam quanto ao tratamento de herança múltipla:
  - Algumas linguagens permitem a herança múltipla de classes, outras não.
- O Java reside em algum lugar intermediário:
  - Ele não permite herança múltipla de classes;
  - Mas fornece uma outra construção, chamada de “interface”, que permite uma forma limitada de herança múltipla.

INTERFACES

- Uma **interface** em Java é uma especificação de um tipo (na forma de um nome de tipo e um conjunto de métodos) que não define nenhuma implementação para os métodos.

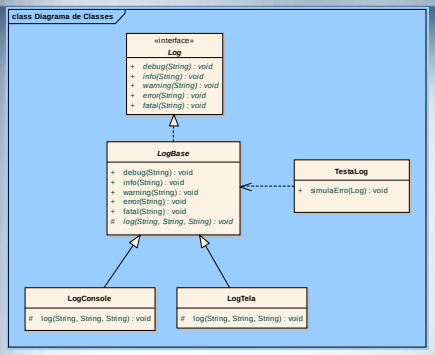
INTERFACES EM JAVA

- A palavra-chave ***interface*** é utilizada em vez de ***class*** no cabeçalho da declaração;
- Todos os métodos em uma interface são abstratos;
- Nenhum corpo de método é permitido;
- A palavra-chave **abstract** não é necessária;
- As interfaces não contêm construtores.

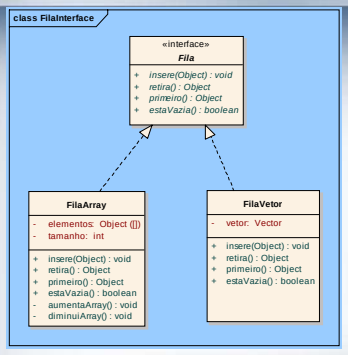
INTERFACES EM JAVA

- Todas as assinaturas de método em uma interface têm **visibilidade pública**:
  - A visibilidade não precisa ser declarada;
- Apenas **campos constantes** (públicos, estáticos e finais) são permitidos em uma interface:
  - As palavras-chave ***public***, ***static*** e ***final*** podem ser omitidas.

EXEMPLO LOG



EXEMPLO FILA



INTERFACES EM JAVA

- Java permite que qualquer classe estenda no máximo uma outra classe;
- Entretanto, ele permite que uma classe implemente qualquer número de interfaces (além possivelmente de estender uma classe).

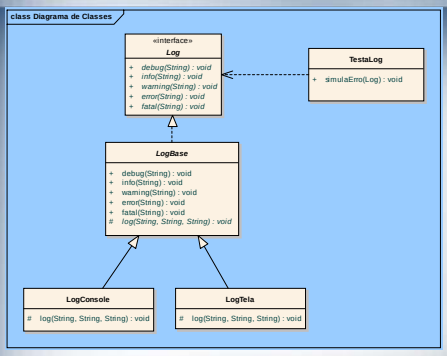
INTERFACES COMO TIPOS

- Quando uma classe implementa uma interface, ela não herda implementação dela, uma vez que as interfaces não podem conter os corpos de métodos;
- Quando introduzimos a herança, enfatizamos dois grandes benefícios da herança:
  - A subclasse herda o código da superclasse.
    - Isto permite reutilizar o código existente e evitar a duplicação de código.
  - A subclasse torna-se um subtipo da superclasse.
    - Isto permite variáveis polimórficas e chamadas de método.

INTERFACES COMO TIPOS

- As interfaces não fornecem o primeiro benefício (uma vez que elas não contêm implementações), mas fornecem o segundo;
- Uma interface define um tipo da mesma maneira como uma classe;
- Isto significa que as variáveis podem ser declaradas como tipos de interface, mesmo que nenhum objeto desse tipo possa existir (somente subtipos);
- As interfaces não podem ter nenhuma instância direta, mas elas servem como supertipo para instâncias de outras classes.

EXEMPLO LOG



INTERFACE COMPARABLE

- Java contém vários operadores de comparação (<, <=, >, >=, ==, !=) que permitem comparar tipos primitivos;
- Estes operadores não podem ser usados para comparar objetos;
- A igualdade de objetos pode ser definida com o método ***equals()***.

INTERFACE COMPARABLE

- A interface **Comparable** é utilizada para comparar objetos entre si;
- O método **compareTo()** deve retornar:
  - Um valor inteiro negativo se objeto em que é invocado é menor do que o objeto passado como parâmetro;
  - Um zero se os dois objetos forem iguais;
  - Um valor inteiro positivo se o objeto em que é invocado é maior do que o objeto passado como parâmetro;