Herança

Aspectos básicos

- Herança é um dos três princípios básicos da programação orientada a objetos, porque permite a criação de classificações hierárquicas.
- Usando herança, você pode criar uma classe geral que defina características comuns a um conjunto de itens relacionados.
- Essa classe poderá então ser herdada por outras classes mais específicas, cada uma adicionando suas características exclusivas.

Aspectos básicos

- No jargão Java, a classe que é herdada se chama *superclasse*. A classe que herda se chama *subclasse*.
- Portanto, uma subclasse é uma versão especializada da superclasse.
- Ela herda todas as variáveis e métodos definidos pela superclasse e adiciona seus próprios elementos exclusivos.
- Java dá suporte à herança, permitindo que uma classe incorpore outra em sua declaração. Isso é feito com a palavra-chave **extends**.
- Exercício da pg. 226

Aspectos básicos

- Ser a superclasse de uma subclasse não significa não poder ser usada separadamente.
- Você só pode especificar uma única superclasse para qualquer subclasse que criar.
- Java não dá suporte a herança de várias superclasses na mesma subclasse.
- No entanto, você pode criar uma hierarquia de herança em que uma subclasse passa a ser a superclasse de outra subclasse.

Acesso a membros e a herança

- Uma grande vantagem de herança é que, uma vez que você tenha criado uma superclasse que defina os atributos comuns a um conjunto de objetos, ela poderá ser usada para criar qualquer número de subclasses mais específicas.
- Com frequência a variável de instância de uma classe é declarada como **private** para não poder ser usada sem autorização ou adulterada.
- Herdar uma classe *não* invalida a restrição de acesso **private**. Logo, ainda que uma subclasse inclua todos os membros de sua superclasse, não poderá acessar os membros declarados como **private.**

Construtores e herança

- Em uma hierarquia, é possível que tanto as superclasses quanto as subclasses tenham seus próprios construtores.
- Isso levanta uma questão importante: que construtor é responsável pela construção de um objeto da subclasse – o da superclasse, o da subclasse ou ambos?
- A resposta é esta: o construtor da superclasse constrói a parte do objeto referente à superclasse e o construtor da subclasse constrói a parte da subclasse. Na prática, porém, a maioria das classe terá construtores explícitos.
- Exercício da pg. 232

Construtores e herança

- Quando tanto a superclasse quanto a subclasse definem o construtor, o processo é um pouco mais complicado, porque os dois construtores devem ser executados.
- Nesse caso, você deve usar outra das palavras-chave Java, **super**, que tem duas formas gerais.
- A primeira chama um construtor da superclasse. A segunda é usada para acessar um membro da superclasse ocultado pelo membro de uma subclasse.
- Aqui examinaremos o seu primeiro uso.

Construtores e herança

- Usando uma superclasse para chamar construtores.
- Uma subclasse pode chamar um construtor definido por sua superclasse usando a forma de **super** a seguir:

super(*lista-parâmetros*);

- *Lista-parâmetros* especifica qualquer parâmetro requerido pelo construtor na superclasse.
- A primeira instrução executada dentro do construtor de uma subclasse deve sempre ser **super().**
- Exercício da pg. 234.

Construtores e herança

- Agora veremos como usar super para acessar membros da superclasse.
- Há uma segunda forma de **super** que age um pouco como **this**, exceto por referenciar sempre a superclasse da subclasse em que é usada.
- Essa aplicação tem a forma geral a seguir:

super.membro

- Aqui, *membro* pode ser um método ou uma variável de instância.
- Exercício da pg. 239