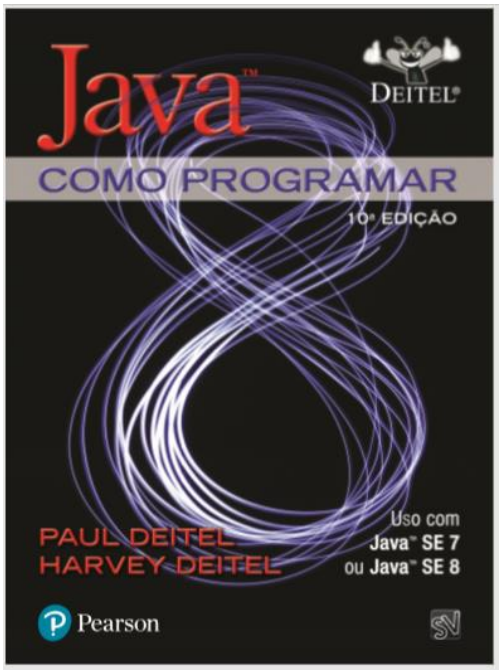
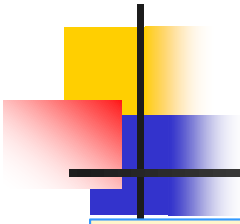


Java

Notas de Aula

Prof. André Bernardi

andrebernardi@unifei.edu.br

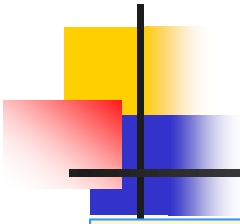


Revisão de POO

- Cada classe que você cria se torna um novo tipo que pode ser usado para declarar variáveis e criar objetos.
- Você pode declarar novas classes conforme necessário. Essa é uma das razões pelas quais o Java é conhecido como uma linguagem extensível.

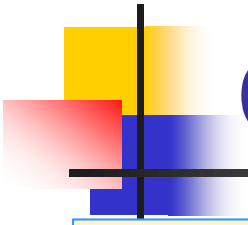
Classes

- **Toda** declaração de classe que começa com o modificador de acesso ***public*** deve ser armazenada em um arquivo que tenha o **mesmo** nome da classe e termina com a extensão de nome de arquivo .java .
- Cada declaração de classe contém a palavra reservada **class** seguida imediatamente pelo nome da classe.
- Nomes de classes, métodos e de variáveis são identificadores. Por convenção, todos usam nomes padronizados na notação camelo. Os nomes de classes começam com uma letra maiúscula e os nomes de métodos e variáveis começam com uma letra minúscula.



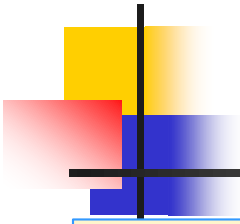
Classes

- Um objeto possui atributos que são implementados como **variáveis de instâncias** e carregados com ele ao longo de sua existência.
- Variáveis de instância existem antes que os métodos sejam chamados em um objeto, enquanto os métodos estão em execução e após a execução deles.
- Uma classe normalmente contém um ou mais métodos que manipulam as variáveis de instância que pertencem a objetos específicos da classe.



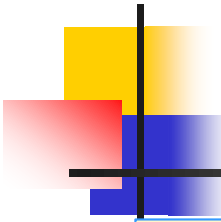
Classes

- As variáveis de instância são declaradas dentro de uma declaração de classe, mas fora dos corpos das declarações de método da classe.
- Cada objeto (instância) da classe tem sua própria cópia de cada uma das variáveis de instância da classe.
- A maioria das declarações de variáveis de instância são precedidas pela palavra-reservada **private**, que é um modificador de acesso. Variáveis ou métodos declarados com modificador de acesso **private** são acessíveis apenas para métodos da classe em que são declarados.



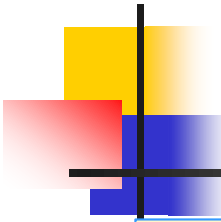
Classes

- Os parâmetros são declarados em uma lista de parâmetros separados por vírgula, localizada dentro dos parênteses que seguem o nome do método na declaração do método. Múltiplos parâmetros são separados por vírgulas. Cada parâmetro deve especificar um tipo seguido por um nome de variável.
- As variáveis declaradas no corpo de um determinado método são **variáveis locais** e podem ser usadas apenas nesse método. Quando um método termina, os valores de suas variáveis locais são perdidos. Os **parâmetros** de um método são variáveis locais do método.



Classes

- O corpo de cada método é delimitado pelas chaves esquerda e direita (**{** e **}**).
- O corpo de cada método contém uma ou mais instruções que realizam a(s) tarefa(s) do método.
- O tipo de retorno do método especifica o tipo de dados retornados para o chamador do método. A palavra-chave **void** indica que um método executará sua tarefa, mas não retornará nenhuma informação.
- Parênteses vazios após um nome de método indicam que o método não requer nenhum parâmetro para executar sua tarefa.



Classes

- Quando um método que especifica um tipo de retorno diferente de **void** é chamado e completa sua tarefa, o método deve retornar um resultado para seu método de chamada.
- A declaração de retorno (**return**) passa um valor de um método chamado de volta para seu chamador.
- As classes geralmente fornecem métodos públicos (**public**) para permitir que os clientes da classe ajustem ou obtenham variáveis de instância privativas (**private**).

Exemplo

```
1   // Figura 3.1: Account.java
2   // Classe Account que contém uma variável de instância name
3   // e métodos para configurar e obter seu valor.
4
5   public class Account
6   {
7      private String name; // variável de instância
8
9      // método para definir o nome no objeto
10     public void setName(String name)
11     {
12        this.name = name; // armazena o nome
13     }
14
15     // método para recuperar o nome do objeto
16     public String getName()
17     {
18        return name; // retorna valor do nome para o chamador
19     }
20  } // fim da classe Account
```

Figura 3.1 | A classe Account que contém uma variável de instância name e métodos para *configurar* e *obter* seu valor.

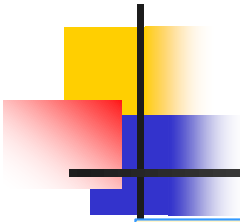
```
Initial name is: null
Please enter the name:
Jane Green
Name in object myAccount is:
Jane Green
```

```java
// Figura 3.2: AccountTest.Java
// Cria e manipula um objeto Account.
import java.util.Scanner;

public class AccountTest
{
   public static void main(String[] args)
   {
      // cria um objeto Scanner para obter entrada a partir da janela de comando
      Scanner input = new Scanner(System.in);

      // cria um objeto Account e o atribui a myAccount
      Account myAccount = new Account();

      // exibe o valor inicial do nome (null)
      System.out.printf("Initial name is: %s%n%n", myAccount.getName());

      // solicita e lê o nome
      System.out.println("Please enter the name:");
      String theName = input.nextLine(); // lê uma linha de texto
      myAccount.setName(theName); // insere theName em myAccount
      System.out.println(); // gera saída de uma linha em branco

      // exibe o nome armazenado no objeto myAccount
      System.out.printf("Name in object myAccount is:%n%s%n",
         myAccount.getName());
   }
} // fim da classe AccountTest
```

Figura 3.2 | Criando e manipulando um objeto Account.

Usando objetos

- Uma classe que cria um objeto de outra classe e, em seguida, chama os métodos desse objeto, é chamada de classe **driver**.
- Uma expressão de criação de instância de classe começa com a palavra-reservada **new** e cria um novo objeto.
- O método da classe *Scanner* **nextLine** lê os caracteres até que um caractere de nova linha seja encontrado e retorna os caracteres como uma String.



Usando objetos

- Um **construtor** é semelhante a um método, mas é chamado implicitamente pelo operador **new** para inicializar as variáveis de instância de um objeto no momento em que o objeto é criado.
- Para chamar um método de um objeto, siga o nome do objeto com um operador ponto, o nome do método e um bloco de parênteses contendo os argumentos do método.
- Variáveis locais não são inicializadas automaticamente.

Usando objetos

- Cada variável de instância tem um valor inicial padrão, um valor fornecido pelo **Java** quando você não especifica o valor inicial da variável da instância.
- O valor padrão para uma variável de instância do tipo **String** é `null`.
- Uma chamada de método fornece valores - conhecidos como argumentos - para cada um dos parâmetros do método. O valor de cada argumento é atribuído ao parâmetro correspondente no cabeçalho do método.

Usando objetos

- O número de argumentos em uma chamada de método deve corresponder ao número de parâmetros na lista de parâmetros da declaração do método.
- Um argumento é o valor real de um parâmetro de método.
- Os tipos de argumento na chamada do método devem ser consistentes com os tipos dos parâmetros correspondentes na declaração do método.

Compilação

Compilando e executando um aplicativo com várias classes

- O comando **javac** pode compilar várias classes de uma só vez. Basta listar os nomes dos arquivos de código fonte após o comando, com cada nome de arquivo separado por um espaço a partir do próximo. Se o diretório que contém o aplicativo incluir apenas os arquivos de um aplicativo, você poderá compilar todas as suas classes com o comando ***javac *.java***. O asterisco (*) em *.java indica que todos os arquivos no diretório atual que terminam com a extensão de arquivo ".java" devem ser compilados.

Compilação

Para o exemplo da figura 3.1, 3.2 do livro do Deitel, podemos compilar:

```
javac Account.java AccountTest.java
```

ou:

```
javac *.java
```

Para executar, usamos o comando:

```
java AccountTest
```

Compilação e execução

- Em um aplicativo, você deve chamar a maioria dos métodos, exceto **main**, explicitamente para instruí-los a fazer suas tarefas.
- Uma parte essencial da ativação da JVM para localizar e chamar o método **main** para iniciar a execução do aplicativo é a palavra reservada **static**, que indica que ***main*** é um método estático que pode ser chamado sem primeiro criar um objeto da classe na qual o método é declarado.

Compilação e execução

- Classes compiladas em um mesmo diretório tem um relacionamento especial. Por padrão, essas classes são consideradas como sendo do mesmo **pacote** — conhecido como pacote padrão. As classes do mesmo pacote são importadas **implicitamente** para os arquivos de código-fonte de outras classes desse mesmo pacote.
- Uma declaração **import** não é necessária quando uma classe utiliza outra no mesmo pacote.

Compilação e execução

- Uma declaração **import**, também não é necessária se você sempre se referir a uma classe com um nome **totalmente qualificado**, que inclui o nome do pacote e o nome da classe.
- A maioria das classes que pertencem as bibliotecas e você utilizará nos programas Java, precisam ser importadas explicitamente.

Tipos Primitivos **X** Tipos por Referência

- Tipos no Java são divididos em duas categorias — **primitivos** e por **referência**. Os tipos primitivos são `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` e `double`. Todos os outros são por referência; portanto, classes, que especificam os tipos de objeto, são tipos por referência.
- As variáveis de instância de tipo **primitivo** são inicializadas por padrão. Variáveis dos tipos `byte`, `char`, `short`, `int`, `long`, `float` e `double` são inicializadas como 0. As variáveis de tipo `boolean` são inicializadas como false.

Tipos Primitivos **X** Tipos por Referência

- Uma variável de tipo primitivo pode armazenar exatamente **um valor** de seu tipo declarado por vez.
- Variáveis de tipo por referência (chamadas referências) armazenam o local de um objeto na memória do computador. Essas variáveis referenciam objetos no programa. O objeto que é referenciado pode conter muitas variáveis de instância e métodos. Uma referência a um objeto é necessária para chamar os métodos de um objeto.
- Uma variável de tipo primitivo não referencia um objeto e, portanto, não pode ser utilizada para invocar um método.

Números de ponto flutuante

- Um número de ponto flutuante é um número com um ponto decimal. Java fornece **dois** tipos primitivos para armazenar números de ponto flutuante na memória - `float` e `double`.
- Variáveis do tipo `float` representam números de ponto flutuante de precisão simples e possuem 7 dígitos significativos. Variáveis do tipo `double` representam números de ponto flutuante de precisão dupla. Eles exigem duas vezes mais memória que as variáveis `float` e fornecem 15 dígitos significativos - aproximadamente o dobro da precisão das variáveis `float`.

Números de ponto flutuante

- Literais de ponto flutuante são do tipo `double` por padrão.
- O método `nextDouble` de Scanner retorna um valor `double`.
- O especificador de formato `%f` é utilizado para gerar saída de valores de tipo `float` ou `double`. Já o especificador de formato `%.2f` especifica que dois dígitos da precisão devem ser gerados à direita do ponto decimal no número de ponto flutuante.
- O valor padrão para uma variável de instância do tipo `double` é `0.0`, e o valor padrão para uma variável de instância do tipo `int` é `0`.

Representando uma classe em UML

```
// Figura 3.1: Account.java
// Classe Account que contém uma variável de instância name
// e métodos para configurar e obter seu valor.
public class Account{
   private String name; // variável de instância
   // método para definir o nome no objeto
   public void setName(String name){
       this.name = name; // armazena o nome
   }
   // método para recuperar o nome do objeto
   public String getName(){
       return name; // retorna valor do nome para o chamador
   }
} // fim da classe Account
```

Representando uma classe em UML

- Na **UML**, cada classe é modelada em um diagrama de classe como um retângulo com três compartimentos. O compartimento superior contém o **nome da classe** centralizado horizontalmente em negrito. O compartimento do meio exibe os **atributos** da classe, que correspondem às variáveis de instância em Java. O inferior inclui as operações da classe, que correspondem a **métodos** e **construtores** em Java.
- A **UML** representa variáveis de instância como um nome de atributo, seguido por dois-pontos e o tipo.
- Os atributos **privados** são precedidos por um sinal de subtração (–) na **UML**.

Representando uma classe em UML

- A **UML** modela operações listando o nome delas seguido por um conjunto de parênteses. Um sinal de adição (+) na frente do nome da operação indica que é uma do tipo **public** na UML (isto é, um método *public* em Java).
- A **UML** modela um parâmetro de uma operação listando o nome dele, seguido por um caractere de dois-pontos e o tipo dele entre parênteses depois do nome de operação.
- A **UML** indica o tipo de retorno de uma operação colocando dois-pontos e ele depois dos parênteses que se seguem ao nome da operação.

Representando uma classe em UML

- Os diagramas de classe **UML** não especificam tipos de retorno para operações que não retornam valores.
- Declarar variáveis de instância *private* é conhecido como ocultar dados ou informações.

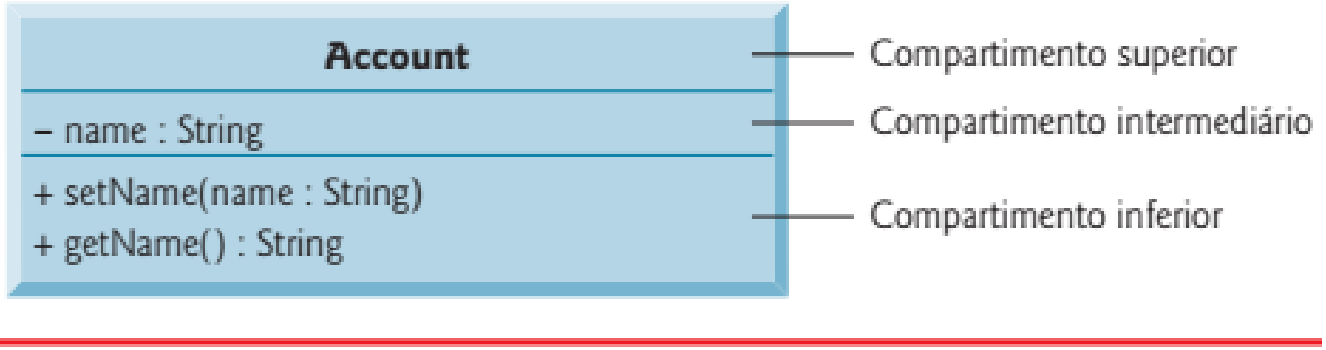
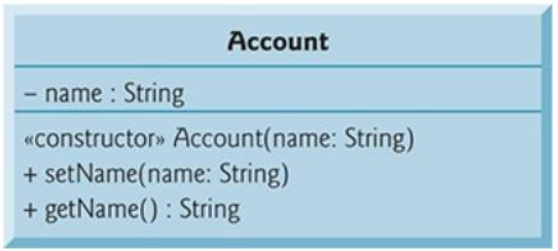


Figura 3.3 | Diagrama UML de classe para a Account da Figura 3.1.

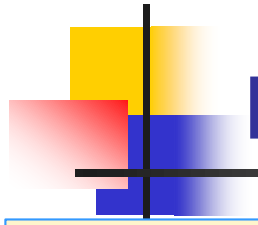
Inicializando objetos com construtores

- As variáveis de instância de tipo por **referência** são inicializadas por padrão como valor `null`.
- Cada classe que você declara pode opcionalmente fornecer um ou mais construtores com parâmetros que podem ser usados para inicializar um objeto de uma classe quando o objeto é criado.
- Java requer uma chamada de **construtor** para todos os objetos criados.
- Os construtores podem especificar parâmetros, mas não retornar tipos.



Inicializando objetos com construtores

- Se uma classe não define construtores, o compilador fornece um **construtor padrão** sem parâmetros e as variáveis de instância da classe são inicializadas com seus valores padrão.
- Se você declarar um construtor para uma classe, o compilador não criará um construtor padrão para essa classe.
- A **UML** modela os construtores no terceiro compartimento de um diagrama de classe. Para distinguir entre um construtor e operações de uma classe, a UML coloca a palavra "*constructor*" entre aspas francesas (« e ») antes do nome do construtor.



Exercício

1) Determine se cada uma das seguintes sentenças é verdadeira ou falsa. Se falsa, explique por quê.

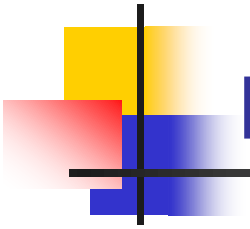
a) Por convenção, os nomes de método são iniciados com letra maiúscula, e todas as palavras subsequentes a ele também começam com letra maiúscula.

b) Uma declaração import não é necessária quando uma classe em um pacote utiliza outra no mesmo pacote.

c) Parênteses vazios que se seguem a um nome de método em uma declaração indicam que ele não requer nenhum parâmetro para realizar sua tarefa.

d) Uma variável de tipo primitivo pode ser utilizada para invocar um método.

e) As variáveis declaradas no corpo de um método private são conhecidas como variáveis de instância e podem ser utilizadas em todos os métodos da classe.



Exercício

f) O corpo de todos os métodos é delimitado pelas chaves esquerda e direita ({ e }).

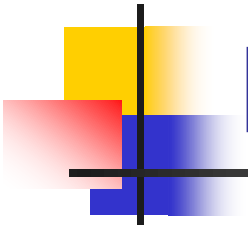
g) As variáveis locais de tipo primitivo são inicializadas por padrão.

h) As variáveis de instância de tipo por referência são inicializadas por padrão com o valor null.

i) Qualquer classe que contém public static void main(String[] args) pode ser usada para executar um aplicativo.

j) O número de argumentos na chamada de método deve corresponder ao de itens na lista de parâmetros da declaração desse método.

k) Os valores de ponto flutuante que aparecem no código-fonte são conhecidos como literais de ponto flutuante e são tipos float por padrão.



Referências

- Java How to Program 3, 4, 5, 6, 7, 8, 9, 10 ed. - Paul Deitel and Harvey Deitel.
- Sun (http://java.sun.com)
- Oracle (http://www.oracle.com/technetwork/java)