Classes e objetos

PCII - Programação Orientada a Objetos em Java

FEG - UNESP - 2021

Sumário

1 Classes

2 Métodos

3 Variáveis

4 Objetos

5 Parâmetros

6 Variáveis de instância

7 Construtores

8 Exemplo adicional

Classes

Declaração de classes

A declaração de uma classe deve possuir os seguintes componentes:

- Modificador de acesso: visibilidade da classe (`public`, `protected` , `private`).
- Identificação: o nome da classe.
- Herança: o nome da superclasse, se houver, precedida da palavra-chave `extends`.
- Interfaces: a lista das interfaces implementadas (se houver), separadas por vírgulas, precedida pela palavra-chave `implements`.
- Corpo da classe: envolto por chaves `{}`, contém os atributos, construtores e métodos da classe.

```
// declaração de classe
<modificador> class MyClass <extends MySuperClass> <implements MyInterface1, ..., MyInterfaceN> {
    // atributos , construtores e métodos
}
```

Classes

Declaração de classes

- A classe `GradeBook` abaixo possui visibilidade pública, não herda de nenhuma classe, não implementa nenhuma interface e contém somente um método `displayMessage()`.
- Há dois possíveis níveis de acesso onde as classes podem ser declaradas:
  - Nível externo (top-level) – classes não-contidas em outra classe.
  - Nível interno (member level) – classes declaradas no corpo de outra classe.

```
// GradeBook.java
public class GradeBook {
   // exibe uma mensagem de boas—vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

} // fim classe GradeBook
```

Classes

Declaração de classes top-level

- Os modificadores de acesso possíveis para uma classe *top-level* são `public` ou *package-private* (sem modificador de acesso).
- Uma classe *top-level* `public` é visível por qualquer outra classe de seu aplicativo.
- Uma classe *top-level* `public` deve estar contida em um arquivo `.java` com o mesmo nome da classe.
- Declarar mais de uma classe *top-level* com o modificador de acesso `public`no mesmo arquivo consiste em um erro de compilação.

```
// GradeBook.java
// classe top-level
public class GradeBook {
   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

} // fim classe GradeBook
```

Classes

Declaração de classes top-level

- Se uma classe *top-level* não possui um modificador de acesso, sua visibilidade é denotada por package-private. Isso significa que essa classe é visível somente dentro de seu pacote.
- Um pacote consiste em um grupo de classes relacionadas contidas em uma mesma pasta (diretório do sistema de arquivos).

```
// GradeBook.java
// classe top-level
public class GradeBook {
   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

} // fim classe GradeBook
```

Classes

Declaração de classes member level

- Classes declaradas internamente a outras classes possuem visibilidades adicionais além de `public` e *package-private*. São elas:
  - `protected`: a classe interna é acessível por qualquer classe definida no pacote onde ela se encontra ou por uma subclasse que herda da classe top-level que a contém.
  - `private`: a classe interna é acessível somente dentro da classe top-level que a contém.

```
// GradeBook.java
// classe top-level
public class GradeBook {
   // exibe uma mensagem de boas—vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

   // classe member level, interna à classe GradeBook
   private class Professor {
      // exibe o nome do professor
      public void displayName() {
            System.out.println("Prof. José F. Vizcaino");
      } // fim método displayName
   } // fim classe Professor

} // fim classe GradeBook
```

Classes

Modificadores de acesso

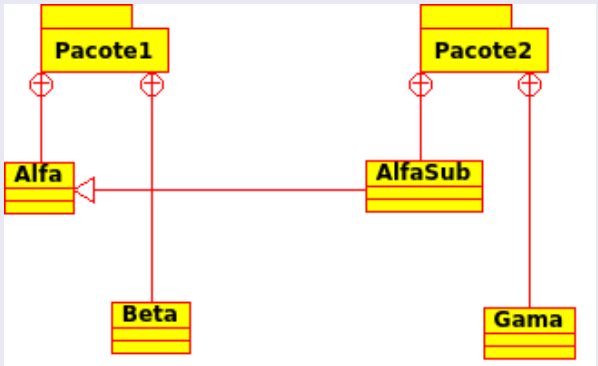
- A tabela a seguir mostra os possíveis modificadores de acesso que podem ser aplicados a classes, métodos e atributos.
- Quando um programador utiliza códigos desenvolvidos por terceiros, os modificadores de acesso desse código definem quais atributos e métodos estarão acessíveis para as novas classes que o programador irá criar.
- Do mesmo modo, ao desenvolver uma nova classe, o programador deve decidir qual a visibilidade dos atributos e métodos de suas classes, antecipando que essas classes serão utilizadas por terceiros.

Modificador	Classe	Pacote	Subclasse	Mundo
`public`	SIM	SIM	SIM	SIM
`protected`	SIM	SIM	SIM	NÃO
nenhum *(package-private)*	SIM	SIM	NÃO	NÃO
`private`	SIM	NÃO	NÃO	NÃO

Classes

Modificadores de acesso

- A figura a seguir mostra quatro classes (`Alfa`, `Beta`, `Gama` e `AlfaSub`) e suas relações (note que a classe `AlfaSub` é subclasse de `Alfa`).



- A tabela a seguir mostra a visibilidade de `Alfa` para as outras classes de acordo com o modificador de acesso utilizado.
- Os modificadores de acesso também podem ser aplicados aos atributos e métodos da classe, produzindo o mesmo efeito para suas visibilidades.

	Visível para			
Modificador de acesso de `Alfa`	`Alfa`	`Beta`	`AlfaSub`	`Gama`
`public`	SIM	SIM	SIM	SIM
`protected`	SIM	SIM	SIM	NÃO
nenhum *(package-private)*	SIM	SIM	NÃO	NÃO
`private`	SIM	NÃO	NÃO	NÃO

Métodos

Declaração de métodos

A declaração de um método possui os seguintes componentes:

- Modificadores: tratam da visibilidade do método (`public`, `protected`, `private`) e se ele não pertence ao objeto, mas à classe (`static`).
- Tipo de retorno: o tipo de dado (primitivo ou referenciado) que o método deverá retornar, ou `void` se o método não retorna informação.
- Identificação: o nome do método.
- Lista de parâmetros: dentro dos parênteses, separados por vírgula e precedidos pelos seus tipos. Se não houver parâmetros, basta utilizar os parênteses vazios `()`.
- Corpo do método: delimitado por chaves `{}`, contém a declaração das variáveis locais e o código que implementa a funcionalidade do método.

```
// declaração de método
<modificadores> <tipo de retorno> myMethod (<tipo> myParameter1, ..., <tipo> myParameterN) {
    // variáveis locais e funcionalidades
}
```

Métodos

Declaração de métodos

- A classe `GradeBook` possui um único método de nome `displayMessage`
- Trata-se de um método com visibilidade pública, não retorna nenhum informação (`void`), não possui parâmetros e nem variáveis locais, sua única função é imprimir na tela **"Bem-vindo ao curso!".**

```
// GradeBook.java
public class GradeBook {
   // exibe uma mensagem de boas—vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

} // fim classe GradeBook
```

Variáveis

Declaração de variáveis

A declaração de uma variável contém os seguintes componentes:

- Modificadores: tratam da visibilidade da variável (`public`, `protected`, `private`), se ela pertence à classe (`static`) e se seu valor, uma vez atribuído, não pode ser modificado (`final`).
- Tipo: o tipo (primitivo ou por referência) da variável.
- Identificação: o nome da variável.

Obs: Os modificadores `public`, `protected`, `private` não se aplicam a variáveis locais e parâmetros.

```
// declaração de variável
<modificadores> <tipo> myAttribute;
```

Objetos

Instanciando Objetos

- A classe `GradeBook` trata-se apenas de uma receita para a criação de um objeto.
- Para imprimir a mensagem **"Bem-vindo ao curso!"** é necessário criar um objeto e executar o método `displayMessage()`.
- Por si só, a classe `GradeBook` não é um aplicativo.

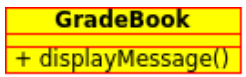


Diagrama UML da classe `GradeBook`.

```
// GradeBook.java
public class GradeBook {
   // exibe uma mensagem de boas—vindas para o usuário
   public void displayMessage() {
      System.out.println("Bem-vindo ao curso!");
   } // fim método displayMessage

} // fim classe GradeBook
```

Objetos

Instanciando Objetos

- Ao tentar executar a classe `GradeBook` haverá um erro de compilação:

```
Error: Main method not found in class GradeBook, please define the main method as:
       public static void main(String[] args)
       or a JavaFX application class must extend javafx.application.Application
```

- Para corrigir esse problema, deve-se acrescentar um método `main` na classe `GradeBook`.
- Alternativamente, é possível construir uma driver class, ou seja, uma classe separada que contém um método `main` que instancia um objeto da classe que desejamos executar.

```
// GradeBookDriver.java
// Cria um objeto GradeBook e chama seu método displayMessage.
public class GradeBookDriver {
    public static void main( String args[] ) {

        // cria um objeto GradeBook e o atribui à variável myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // chama método displayMessage do objeto myGradeBook
        myGradeBook.displayMessage();

    } // fim do main
} // fim de GradeBookDriver
```

Parâmetros

Adicionando parâmetros ao método

- Modificando a classe `GradeBook` para incluir na mensagem de texto o nome de um curso passado como argumento do método `displayMessage()`:

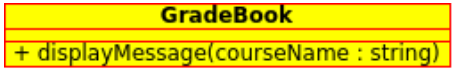


Diagrama UML da classe `GradeBook`.

```
// GradeBook.java
public class GradeBook {
   // exibe uma mensagem de boas—vindas para o usuário
   public void displayMessage(String courseName) {
      System.out.printf("Bem-vindo ao curso %s!", courseName);
   } // fim método displayMessage

} // fim classe GradeBook
```

Parâmetros

Adicionando parâmetros ao método

- O driver `GradeBookDriver` foi atualizado para ler do usuário o nome do curso a ser impresso:

```
// GradeBookDriver.java
// Cria um objeto GradeBook e passa uma String para seu método displayMessage.

import java.util.Scanner; // programa utiliza API Scanner

public class GradeBookDriver {

    public static void main( String args[] ) {
        // cria objeto Scanner para obter dados de entrada do terminal
        Scanner input = new Scanner( System.in );

        // cria um objeto GradeBook e o atribui à variável myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // Imprime e captura dados do terminal
        System.out.println( "Por favor, entre com o nome do curso:" );
        String courseName = input.nextLine(); // captura uma linha de texto

        // chama método displayMessage passando o nome do curso como argumento
        myGradeBook.displayMessage(courseName);

        input.close();
    } // fim do main

} // fim de GradeBookDriver
```

Variáveis de instância

Adicionando atributo do objeto e métodos get e set

- A classe `GradeBook` foi atualizada com um novo atributo `courseName`, que armazena o nome do curso, e dois novos métodos `setCourseName()` e `getCourseName()`.

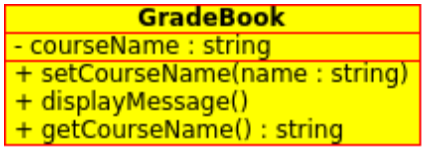


Diagrama UML da classe `GradeBook`.

```
// GradeBook.java
public class GradeBook {
    private String courseName; // nome do curso

    // método para configurar o nome do curso
    public void setCourseName( String name ) {
        courseName = name;
    } // fim método setCourseName

    // método para recuperar o nome do curso
    public String getCourseName() {
        return courseName;
    } // fim método getCourseName

   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.printf("Bem-vindo ao curso %s!", courseName);
   } // fim método displayMessage
} // fim classe GradeBook
```

Variáveis de instância

Adicionando atributo do objeto e métodos get e set

- A declaração de variáveis de instância com o modificador de acesso `private` é o que se chama de ocultamento de dados ou encapsulamento.
- A variável `courseName` foi encapsulada no objeto e só pode ser acessada por métodos do objeto.
- Na classe `GradeBook` foram adicionados os métodos `setCourseName` e `getCourseName` que manipulam a variável `courseName`.

```
// GradeBook.java
public class GradeBook {
    private String courseName; // nome do curso

    // método para configurar o nome do curso
    public void setCourseName( String name ) {
        courseName = name;
    } // fim método setCourseName

    // método para recuperar o nome do curso
    public String getCourseName() {
        return courseName;
    } // fim método getCourseName

   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.printf("Bem-vindo ao curso %s!", courseName);
   } // fim método displayMessage
} // fim classe GradeBook
```

Variáveis de instância

Adicionando atributo do objeto e métodos get e set

- O driver `GradeBookDriver` agora manipula o atributo `courseName` do objeto.

```
// GradeBookDriver.java
// Cria um objeto GradeBook e manipula seu atributo.
import java.util.Scanner; // programa utiliza API Scanner

public class GradeBookDriver
{
    // main method begins program execution
    public static void main( String args[] )    {
        // cria objeto Scanner para obter dados de entrada do terminal
        Scanner input = new Scanner( System.in );

        // cria um objeto GradeBook e o atribui à variável myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // Imprime e captura dados do terminal
        System.out.println( "Por favor, entre com o nome do curso:" );
        String courseName = input.nextLine(); // captura uma linha de texto

        myGradeBook.setCourseName( courseName ); // configura o nome do curso

        // chama método displayMessage passando o nome do curso como argumento
        myGradeBook.displayMessage();
        input.close();
    } // fim do main

} // fim de GradeBookDriver
```

Construtores

Adicionando métodos construtores

- Um construtor é um método especial da classe responsável pela inicialização de um objeto.
- O construtor deve ser um método público e deve ser identificado com o mesmo nome da classe.

```
// GradeBook.java
public class GradeBook {
    private String courseName; // nome do curso

    // construtor inicializa o atributo courseName com um String fornecido como argumento
    public GradeBook( String name ) {
        courseName = name; // inicializa courseName
    } // fim construtor

    // método para configurar o nome do curso
    public void setCourseName( String name ) {
        courseName = name;
    } // fim método setCourseName

    // método para recuperar o nome do curso
    public String getCourseName() {
        return courseName;
    } // fim método getCourseName

   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.printf("Bem-vindo ao curso %s!", courseName);
   } // fim método displayMessage
} // fim classe GradeBook
```

Construtores

Adicionando métodos construtores

- Se não for explicitamente declarado um construtor, o compilador fornece um construtor padrão que configura as variáveis de instância para seus valores padrões que dependem do tipo.

```
// GradeBook.java
public class GradeBook {
    private String courseName; // nome do curso

    // construtor inicializa o atributo courseName com um String fornecido como argumento
    public GradeBook( String name ) {
        courseName = name; // inicializa courseName
    } // fim construtor

    // método para configurar o nome do curso
    public void setCourseName( String name ) {
        courseName = name;
    } // fim método setCourseName

    // método para recuperar o nome do curso
    public String getCourseName() {
        return courseName;
    } // fim método getCourseName

   // exibe uma mensagem de boas-vindas para o usuário
   public void displayMessage() {
      System.out.printf("Bem-vindo ao curso %s!", courseName);
   } // fim método displayMessage
} // fim classe GradeBook
```

Construtores

Adicionando métodos construtores

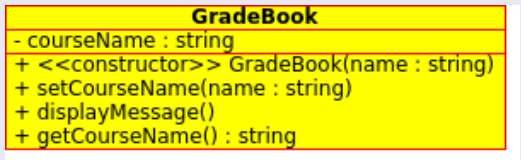


Diagrama UML da classe `GradeBook`.

```
// GradeBook.java
public class GradeBook {
   private String courseName; // nome do curso

   // construtor inicializa o atributo courseName com um String fornecido como argumento
   public GradeBook( String name ) {
      courseName = name; // inicializa courseName
   } // fim construtor

   // método para configurar o nome do curso
   public void setCourseName( String name ) {
      courseName = name;
   } // fim método setCourseName

   // método para recuperar o nome do curso
   public String getCourseName() {
      return courseName;
   } // fim método getCourseName

  // exibe uma mensagem de boas-vindas para o usuário
  public void displayMessage() {
    System.out.printf("Bem-vindo ao curso %s!", courseName);
  } // fim método displayMessage
} // fim classe GradeBook
```

Construtores

Adicionando métodos construtores

- O driver `GradeBookDriver` agora instancia dois objetos `GradeBook` utilizando o construtor.
- A impressão dos nomes dos cursos armazenados nos objetos evidencia que o estado de um objeto é independente do outro objeto de mesmo tipo.

```
// GradeBookDriver.java
// Cria um objeto GradeBook e manipula seu atributo.

public class GradeBookDriver
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // Cria dois objetos GradeBook
        GradeBook gradeBook1 = new GradeBook( "Programação de Computadores I" );
        GradeBook gradeBook2 = new GradeBook( "Programação de Computadores II" );

        // Imprime o nome dos cursos dos dois objetos
        System.out.printf( "nome do curso no objeto gradeBook1 é: %s\n", gradeBook1.getCourseName() );
        System.out.printf( "nome do curso no objeto gradeBook2 é: %s\n", gradeBook2.getCourseName() );

    } // fim main

} // fim classe GradeBookDriver
```

Exemplo adicional

Manipulando uma conta bancária

- A classe `Account` modela uma conta bancária simples que exemplifica os conceitos de declaração de classes, métodos, variáveis e construtores.

```
// Account.java
// Uma classe que abstrai uma conta bancária com um construtor para inicializar o saldo.

public class Account {

   private double balance; // variável de instância que armazena o saldo

   // construtor
   public Account( double initialBalance ) {
      // Inicializa o saldo caso o argumento seja maior do que 0.0;
      // Caso contrário, o saldo é inicializado com o valor default 0.0
      if ( initialBalance > 0.0 )
         balance = initialBalance ;
   } // fim construtor

   // crédito adicionado à conta
   public void credit( double amount ) {
      balance = balance + amount; // adicione o montante ao saldo
   } // fim método credit

   // retorna o saldo da conta
   public double getBalance() {
      return balance;
   } // fim método getBalance

} // fim classe Account
```

Exemplo adicional

Manipulando uma conta bancária

- A classe `AccountDriver` consiste em um *driver* para a classe `Account`.

```
// Cria e manipula um objeto do tipo Account.
import java.util.Scanner;
public class AccountDriver {
   public static void main( String args[] ) {
      Account account1 = new Account( 50.00 ); // cria uma conta bancária
      Account account2 = new Account( -7.53 ); // cria uma conta bancária

      // imprime o saldo inicial de cada conta
      System.out.printf( "Saldo de account1: $%.2f\n", account1.getBalance() );
      System.out.printf( "Saldo de account2: $%.2f\n\n", account2.getBalance() );

      // cria um objeto Scanner para a leitura de dados
      Scanner input = new Scanner( System.in );
      double depositAmount; // montante que será depositado na conta

      System.out.print( "Entre o depósito para account1: " );
      depositAmount = input.nextDouble(); // captura valor do terminal
      account1.credit( depositAmount ); // credita o montante no saldo da primeira conta

      System.out.print( "Entre o depósito para account2:: " );
      depositAmount = input.nextDouble(); // captura valor do terminal
      account2.credit( depositAmount ); // credita o montante no saldo da segunda conta

      // imprime os saldos das duas contas
      System.out.printf( "Saldo de account1: $%.2f\n", account1.getBalance() );
      System.out.printf( "Saldo de account2: $%.2f\n", account2.getBalance() );
      input.close();
   } // fim main
} // fim classe AccountDriver
```

Referências

1. Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 8a. Ed.
2. The Java Tutorials (Oracle)
   http://docs.oracle.com/javase/tutorial/
3. Java Tutorial (w3school)
   https://www.w3schools.com/java/
4. Eckel, B. Thinking in Java. 2. ed.
   http://mindview.net/Books
5. Introduction to Computer Science using Java
   http://chortle.ccsu.edu/java5/index.html