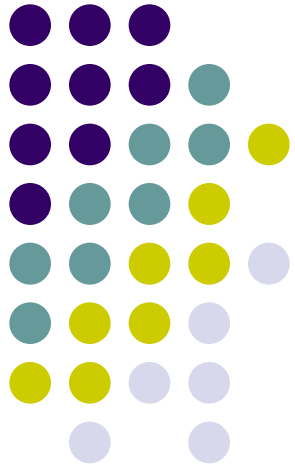
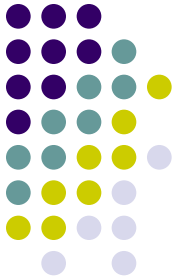
COM220
Aula 9
Construção de Classes

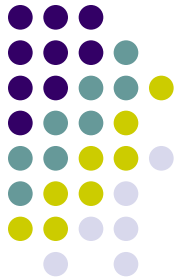
Prof. Laércio Baldochi



Conteúdo



- Prática com classes
  - Herança
  - Métodos *getters* e *setters*
  - Modificadores *static* e *final*
  - Classes concretas e abstratas
  - Associações
  - Métodos construtores

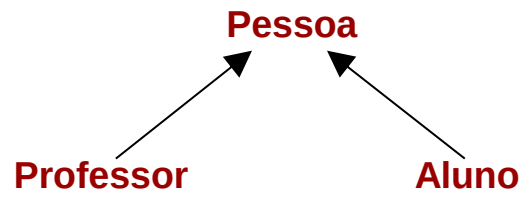


Herança

- Mecanismo que permite a reutilização de código entre classes que possuem características comuns
- Superclasse
  - Classe que fornece atributos e comportamento para outras classes
- Subclasse
  - Especializações de superclasses
  - Produzem instâncias que correspondem a novas versões do modelo original

Herança

```
class Pessoa {
  //atributos
  private String nome, endereco,
                 sexo;
  private int idade;
  //métodos
}

class Professor {
  //atributos
  private String titMaxima;
  //métodos
}
```



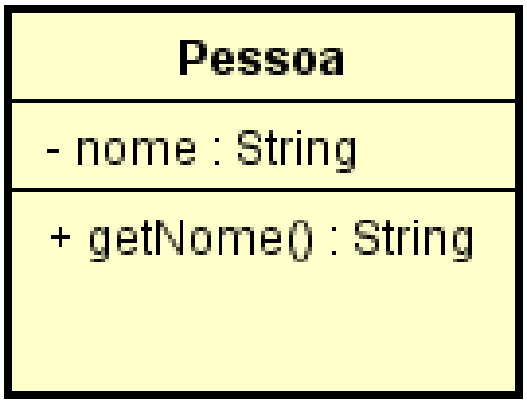

```
class Pessoa {
 //atributos
  private String nome, endereco,
                 sexo;
  private int idade;
  //métodos
}

class Professor extends Pessoa {
   private String titMaxima;
   //métodos da classe Professor
}
```

"extends" é a palavra reservada que indica que uma classe (**subclasse**) está herdando as características de outra classe (**superclasse**)

Getters

São métodos usados para **retornar valores** de atributos de uma **instância**, como por exemplo **retornando o nome** de uma **instância** da classe **Pessoa**:

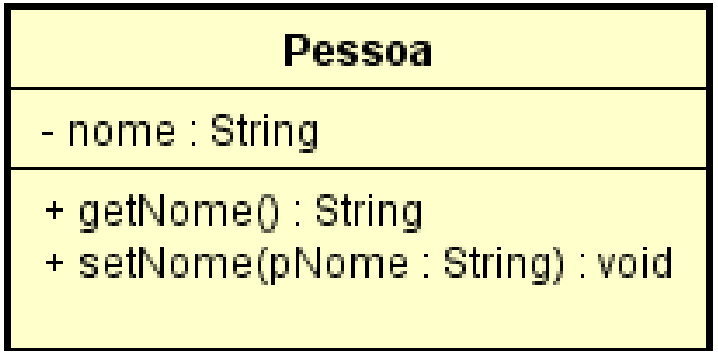
```
public class Pessoa {
        public String nome;

public String getNome() {
    return nome;
    }
}
```



Setters

São métodos usados para **atribuir valores** à atributos de uma **instância**, como por exemplo **atribuindo um valor para a variável nome** de uma **instância** da classe **Pessoa**:

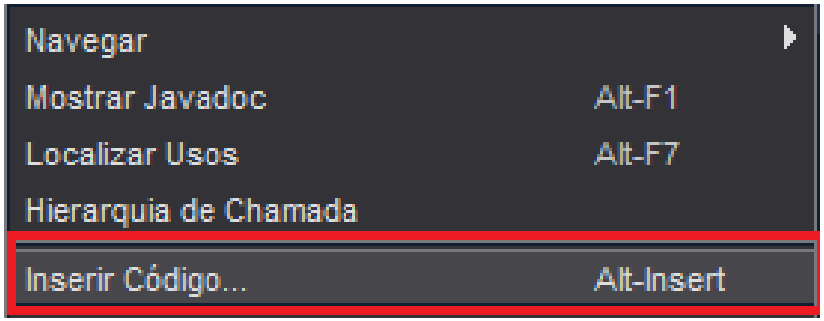
```
public class Pessoa() {
      public String nome;

public void setNome(String pNome){
            nome = pNome;
      }
}
```

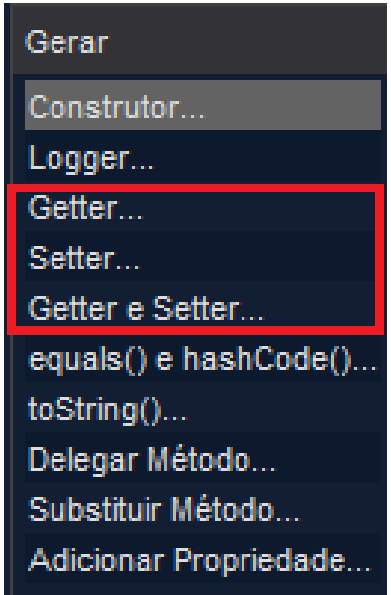


Métodos Setters e Getters

- Devem sempre estar dentro das delimitações da classe, ou seja, precisam estar dentro das chaves {}.

- Na IDE Netbeans é possível gerar os métodos ***setters*** e ***getters*** automaticamente para agilizar o desenvolvimento, basta clicar com o botão direito sem sair das delimitações da classe e clicar em **inserir código**, como mostra a figura abaixo.



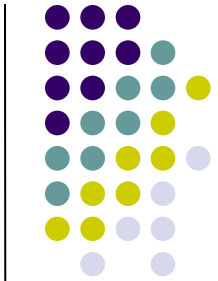


Modificador *Static* para Variáveis

- Variáveis estáticas vão existir sem a dependência da criação de instâncias. São variáveis associadas à classe, então podem ser usadas como no exemplo abaixo:

```
public class Pessoa {
        public static String nome = “Laura”;
}

public class TestaPrograma {

        public static void main(String args[]) {
                System.out.println(Pessoa.nome);
    }
}
```

Modificador *Static* para Métodos

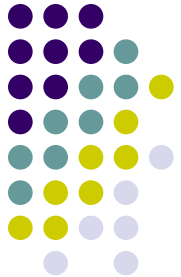


- Métodos estáticos podem ser usados sem a necessidade de criar uma instância, pois eles são associados à classe, como por exemplo:

```
public class Pessoa {
        public static String nome = “Laura”;

public static String getFrase() {
        return “Meu nome é “ +nome;
    }
}
public class TestaPrograma {
        public static void main(String args[]) {
                System.out.println(Pessoa.getFrase());
    }
}
```

- **Nota: Variáveis não estáticas não podem ser referenciadas por métodos estáticos.**

Modificador *Final* para Variáveis

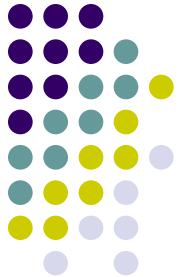


- O modificador ***final*** em variáveis indica que os valores designados à essas variáveis não podem ser mudados;
- Geralmente é usado em conjunto com o modificador ***static*** para tornar uma variável constante;

public static final String NOME = “Pedro”;

- Nas boas práticas de programação, variáveis com modificadores ***static*** e ***final*** devem ser declaradas em maiúsculo.

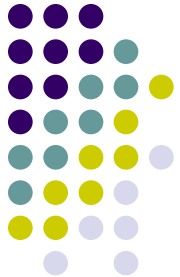
Modificador *Final* para Métodos



- Métodos declarados como ***final*** não podem ser sobrescritos pelas classes filhas que herdam tais métodos de uma superclasse;
- Usar métodos com o modificador ***final*** pode ser útil quando a implementação de uma classe não deve ser mudada de maneira alguma por uma subclasse.

```
public class Pessoa {
        public String nome;

public final void comer() {
        System.out.println("A pessoa está comendo!");
        }
}
```

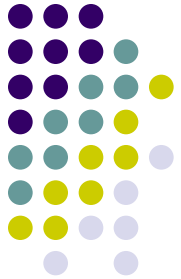
Modificador *Final* para Classes



- Uma classe que foi declarada como ***final*** não pode ser superclasse de outras;
- O modificador ***final*** em classes pode ser útil quando se deseja que uma classe não seja estendida (não tenha subclasses). Ex: classe String

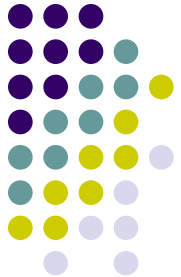
```
public final class Pessoa {
        }
```

Concreta X Abstrata



- Classe concreta
  - Permite que sejam produzidas instâncias (através do operador new)
- Classe abstrata
  - Não permite instanciação, ou seja, não é possível usar o operador ***new*** com classes abstratas.
  - São implementadas com o objetivo de serem estendidas por subclasses concretas
  - Contém atributos e comportamentos comuns a suas subclasses

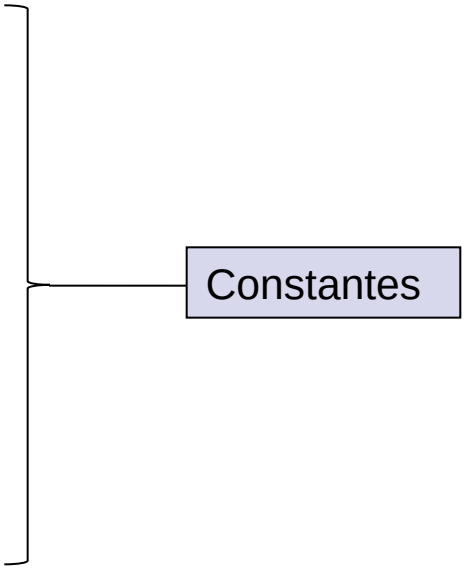
Classe abstrata



- **Classes abstratas** podem conter ou não métodos abstratos, mas se uma classe possui **um método abstrato** pelo menos, então a classe **deve** ser declarada como **abstrata**
- Se uma classe é subclasse(***extends***) de uma classe abstrata, então ela é obrigada a implementar todos os seus métodos abstratos

Classe concreta

```
/**
  Classe de definição de constantes gerais
*/
public class Util {
   public static final String MASCULINO = "M";
   public static final String FEMININO = "F";

   public static final int MIN_IDADE = 10;
   public static final int MAX_IDADE = 120;

   public static final String GRADUACAO = "Graduação";
   public static final String ESPECIALIZACAO = "Especialização";
   public static final String MESTRADO = "Mestrado";
   public static final String DOUTORADO = "Doutorado";

   public static final String CC = "Ciências da Computação";
   public static final String SI = "Sistemas de Informação";
}
```



- Modificador static
  - Indica que um atributo ou método poderá ser acessado sem a criação de uma instância

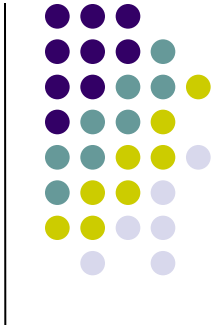
Classe abstrata

```
/**
  Implementação de uma classe abstrata
*/
public abstract class Pessoa {
  private String aNome = "", aEndereco = "", aSexo = "";
  private int aIdade = 0;  //Atributos do tipo int


  public void setNome(String pNome) {
    aNome = pNome;
  }

  public void setEndereco(String pEndereco) {
    aEndereco = pEndereco;
  }

 public void setSexo(String pSexo) {
    if ((!pSexo.equalsIgnoreCase(Util.MASCULINO)) &&
        (!pSexo.equalsIgnoreCase(Util.FEMININO))) {
      System.out.println("O conteúdo informado para"+
        " o atributo sexo não é válido");
    } else {
      aSexo = pSexo;
    }
  }

...
```

Atributos

Métodos de atribuição de informação set...()

Classe abstrata (cont)



```
public void setIdade(int pIdade)  {
    //Determina uma regra de validação do atributo idade,
    //evitando que qualquer valor seja atributo
    if ((pIdade < Util.MIN_IDADE) || (pIdade > Util.MAX_IDADE)) {
      System.out.println("Idade deve ser maior ou igual a "+
                         Util.MIN_IDADE  +" e menor ou igual "+
                         Util.MAX_IDADE);
    } else {
      aIdade = pIdade;
    }
  }

  public String getNome() {   return aNome;  }

  public String getEndereco()  {    return aEndereco;   }

  public String getSexo() {   return aSexo;  }

  public int getIdade() {   return aIdade;  }


  public String getDescricao() {
    String mensagem = aNome+" reside na "+aEndereco+
                      " e possui "+aIdade+" anos de idade";
    if (aSexo.equalsIgnoreCase(Util.MASCULINO))
      return mensagem = "O Sr. "+mensagem;
    else
      return mensagem = "A Sra. "+mensagem;
  }
}
```

Métodos de recuperação de informação get...()

Subclasse

```
/**
  Implementação da classe Professor que é uma subclasse da
  classe Pessoa. Desse modo a classe Pessoa é considerada
  a super classe.
*/
public class Professor extends Pessoa {
  private String aTitMaxima = "";

  //Atribui a titulação máxima
  public void setTitMaxima(String pTitMaxima) {
    //Realiza o controle para verificar se a titulação é válida
    if ((!pTitMaxima.equalsIgnoreCase(Util.GRADUACAO)) &&
        (!pTitMaxima.equalsIgnoreCase(Util.ESPECIALIZACAO)) &&
        (!pTitMaxima.equalsIgnoreCase(Util.MESTRADO)) &&
        (!pTitMaxima.equalsIgnoreCase(Util.DOUTORADO))) {
      System.out.println("A titulação informada não é válida!!!");
    } else {
      aTitMaxima = pTitMaxima;
    }
  }

  //Recupera a titulação máxima
  public String getTitMaxima() {
    return aTitMaxima;
  }

}
```

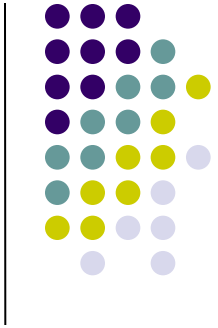
A classe **Professor** é uma subclasse da classe **Pessoa**

Atributos

Método de atribuição de informação set...()

Método de recuperação de informação get...()

Classe de teste



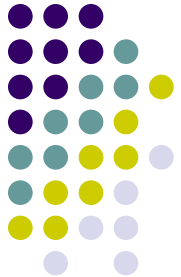
```
/**
  Classe utilizada para teste dos conceitos de OO
*/
public class ExemploClasse {
  public static void main (String par[]) {
     //Cria a instância
     //Primeira intância da classe Professor (objeto 'objProfessor')
     Professor objProfessor = new Professor();

     //Atribui o conteúdo para as variáveis de instância (Atributos)
     objProfessor.setNome("Jose da Silva");
     objProfessor.setEndereco("R das Palmeiras 111");
     objProfessor.setIdade(42);
     objProfessor.setSexo(Util.MASCULINO);
     objProfessor.setTitMaxima(Util.DOUTORADO);

     //Obtém o conteúdo dos atributos Nome, Endereço e Idade e imprime esse
     //conteúdo para a saída padrão utilizando System.out.println();
     System.out.println("Nome: "+objProfessor.getNome());
     System.out.println("Endereço: "+objProfessor.getEndereco());
     System.out.println("Idade: "+objProfessor.getIdade());
     System.out.println("Sexo: "+objProfessor.getSexo());
     System.out.println("Titulação Máxima: "+objProfessor.getTitMaxima());

     //Imprime o conteúdo detalhado através do método getDescricao()
     System.out.println(objProfessor.getDescricao());
  }
}
```

Atributos

Método de atribuição de informação set...()

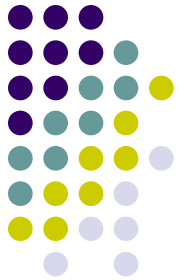
Método de recuperação de informação get...()

Exercício 1



- Crie uma subclasse chamada Aluno que herde as características da classe Pessoa;
- Nessa classe crie um atributo do tipo String para armazenar o nome do curso que o aluno está matriculado;
- Crie também os métodos de atribuição (**set**) e recuperação do curso (**get**);
- No método de atribuição (**set**) deve ser realizado a consistência permitindo que somente sejam aceitos os cursos definidos como constante na classe Util;
- Altere a classe ExemploClasse para que seja criada uma instância de aluno, inicializando os atributos e listando o conteúdo do mesmo.

Relacionamentos



- Um programa OO deve ser capaz de representar as "entidades" presentes em um diagrama de classes
  - Já vimos como representar classes, instâncias, atributos e métodos
- As relações entre classes podem ser representadas por meio de atributos
  - Considere a classe Disciplina
    - **Professores** ministram **Disciplina**
    - **Alunos** cursam **Disciplina**
- Veremos agora como retratar essas relações de associação em nosso programa

Associação

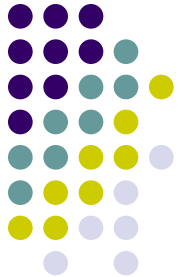
- Devemos, a princípio, definir a classe Disciplina

```
/**
 * Implementação da classe Disciplina
 */
public class Disciplina {
  private String nomeDis;
  private int cargaHoraria;

  public Disciplina(String pnomeDis, int pcargaHoraria) {
    nomeDis = pnomeDis;
    cargaHoraria = pcargaHoraria;
  }

  public String getDescricao() {
    return "  Disciplina: "+nomeDis+"  Carga Horária: "
+cargaHoraria;
  }
}
```

Atributos

Construtor

Método de recuperação de informação get...()



Associação

- Precisamos agora fazer a associação entre as classes
  - Como ambas as classes, Professor e Aluno, possuem associação com a classe Disciplina, podemos colocar o atributo aDisc[] na classe pessoa
  - Evidentemente, se especializarmos novamente a classe Pessoa, criando, a classe TecAdministrativo, não poderíamos colocar o atributo aDisc na superclasse

Associação

- Alterando a classe Pessoa

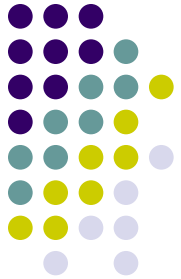
```
/**
  Implementação de uma classe abstrata
*/
public abstract class Pessoa {
  private String aNome = "", aEndereco = "", aSexo = "";
  private int aIdade = 0;
  private Disciplina aDisc[]; //Atributo da classe Disciplina

  public void setDisciplinas(Disciplina pDisc[]) {
    aDisc = pDisc;
  }
  //Conjunto de métodos já definidos
  public String getDisciplina(int pIndice) {
    return aDisc[pIndice].getDescricao();
  }
  public String getDisciplinas() {
    String retorno = "";
    for (int i = 0; i < aDisc.length; i++) {
      retorno += aDisc[i].getDescricao()+"\n";
    }
    return retorno;
  }
}
```

Atributo da classe Disciplina

Método que retorna uma única disciplina utilizando um índice de acesso a **array** aDisc[];

Método que retorna todas as disciplinas constantes no **array** aDisc[];

Exercício 2

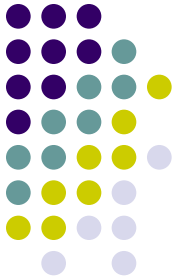


Modifique a classe ExemploClasse realizando a matrícula de um aluno em duas disciplinas

Programação OO

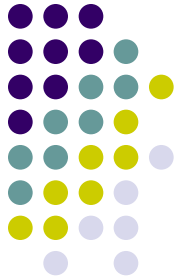
Estrutura de Dados

Construtores



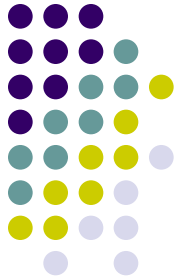
- Toda classe concreta possui um construtor padrão implícito. Esse construtor é utilizado para gerar instâncias da classe quando não há método construtor explícito
- Se o construtor padrão não for suficiente, deve-se declarar explicitamente um ou mais métodos construtores
  - Um método construtor possui o mesmo nome da classe e deve ser public
  - Podem existir vários construtores, entretanto cada definição deve possuir um conjunto único de parâmetros

Construtores (cont)



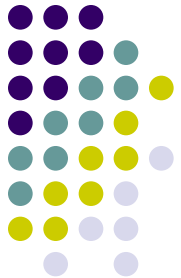
- Caso sejam definidos novos construtores e haja a necessidade de acessar o construtor padrão, esse deve ser definido explicitamente
- Quando um objeto de uma subclasse é instanciado, o construtor da superclasse deve ser chamado para que seja realizada qualquer inicialização nas variáveis de instância da superclasse;
- Os construtores de superclasse não são herdados pela subclasse. Desse modo para acessar determinado construtor da superclasse utiliza-se a referência ***super***.

Definição de construtores



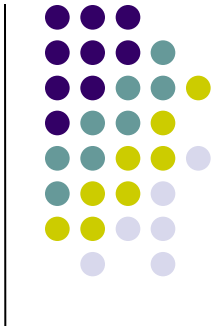
```
class Nome_da_classe {
   //construtor com um parâmetro
   public Nome_da_classe(tipo parâmetro) {
      ....
   }
   //construtor com mais de um parâmetro
   public Nome_da_classe(tipo parâmetro_1, tipo parâmetro_2,...) {
      ....
   }
   //construtor default
   public Nome_da_classe() {
      ....
   }
}
```

Definição de construtores em subclasses



```
class Nome_da_subclasse extends Nome_da_Classe
   {
   //construtor com um parâmetro
   public Nome_da_subclasse([tipo parâmetro]) {
      // escolher 1 dos construtores da superclasse
    ex: super( );
   }
```

Construtores
Uso com this



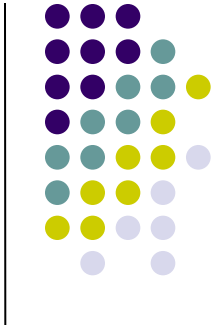
```
public abstract class Pessoa {

  //Definição dos atributos

  //Construtor da classe Pessoa
  public Pessoa(String pNome, String pEndereco,
                String pSexo, int pIdade) {
    aNome = pNome;
    aEndereco = pEndereco;
    aSexo = pSexo;
    aIdade = pIdade;
  }

 //Construtor da classe Pessoa
  public Pessoa(String pNome, String pEndereco, String pSexo,
                int pIdade,  Disciplina pDisc[]) {
    this(pNome, pEndereco, pSexo, pIdade);
    aDisc = pDisc;
  }

  //Construtor default
  public Pessoa() {}

 //Métodos

}
```

Construtores com argumentos

Construtor padrão (default)

A palavra reservada **this** permite referenciar um construtor, um método ou um atributo da classe atual. No caso do construtor permite a reutilização de código, mas no caso de atributos e métodos somente possibilita maior legibilidade no código.

Construtores
Uso com super



```
/**
  Implementação da classe Professor que é uma subclasse da
  classe Pessoa. Desse modo a classe Pessoa é considerada
  a super classe.
*/
public class Professor extends Pessoa {
  private String aTitMaxima = "";

  //Construtor da classe
  public Professor(String pNome, String pEndereco, String pSexo,
                   int pIdade,  String pTitMaxima) {
    super(pNome, pEndereco, pSexo, pIdade);
    aTitMaxima = pTitMaxima;
  }

  //Construtor da classe
  public Professor(String pNome, String pEndereco, String pSexo,
            int pIdade, String pTitMaxima, Disciplina pDisc[]) {
    super(pNome, pEndereco, pSexo, pIdade, pDisc);
    aTitMaxima = pTitMaxima;
  }

  //Construtor default
  public Professor() { }

  //Métodos
}
```

Construtor com argumentos

A palavra reservada **super** permite referenciar um construtor, um método ou um atributo público da superclasse.

Construtor padrão (default)

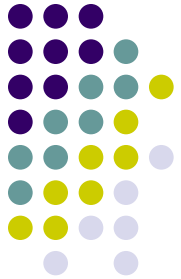
Classe de teste para os novos construtores

```
/**
  Classe utilizada para teste dos conceitos de OO
*/
public class ExemploClasse {
  public static void main (String par[]) {

   //Código anterior

   // Cria array com disciplinas que o Professor Ministra
   Disciplina[] discMinistradas = {
    new Disciplina("Programação Orientada a Objetos", 60),
    new Disciplina("Estruturas de Dados",30)};

   // Cria objeto da classe Professor para guardar todos os dados
   Professor objProfessor = new Professor("Professor 2",
          "R das Oliveiras 45", Util.MASCULINO, 38, Util.DOUTORADO,
          discMinistradas);

   //Imprime o conteúdo detalhado através do método getDescricao()
   System.out.println(objProfessor.getDescricao());



  }
}
```

Cria um array de disciplinas através da instanciação de dois objetos do tipo *Disciplina*

Cria o objeto "objProfessor" utilizando o construtor com argumentos.

Exercício 3



- Implemente o construtor com argumentos para a classe Aluno
- Implemente também o construtor default
- Altere a classe ExemploClasse para que seja criada uma nova instância da classe Aluno através do construtor que possui todos os argumentos, inclusive disciplinas

Créditos

- Os exemplos de código apresentados nessa aula foram elaborados por Roberto Pacheco da UFSC