INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

AULA:

Construtores, Sobrecarga e Encapsulamento (Classes e Objetos)

Programação Orientada a Objetos

Alba Lopes, Profa.

http://docentes.ifrn.edu.br/albalopes
alba.lopes@ifrn.edu.br

Agradecimentos

- *Este curso foi elaborado a partir do material do **ESJUG – Grupo de Usuários Java do Espírito Santo** – Programação Orientada Objetos. Março de 2008.*

Construtores

```
ContaCorrente conta = new ContaCorrente();
```

```
ContaCorrente conta = new ContaCorrente();
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Construtores

```
public class ContaCorrente{
    double saldo;

    //Exemplo de construtor da classe
    ContaCorrente() {
        /* implementação */
    }

    void sacar(double valor){
        saldo = saldo - valor;
    }

}
```

Construtores

```
public class ContaCorrente{
    float saldo;

    //Exemplo de construtor da classe
    ContaCorrente() {
        System.out.println("Nova conta criada");
    }

    boolean sacar(float valor){
        /* implementação */
    }

}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Construtores

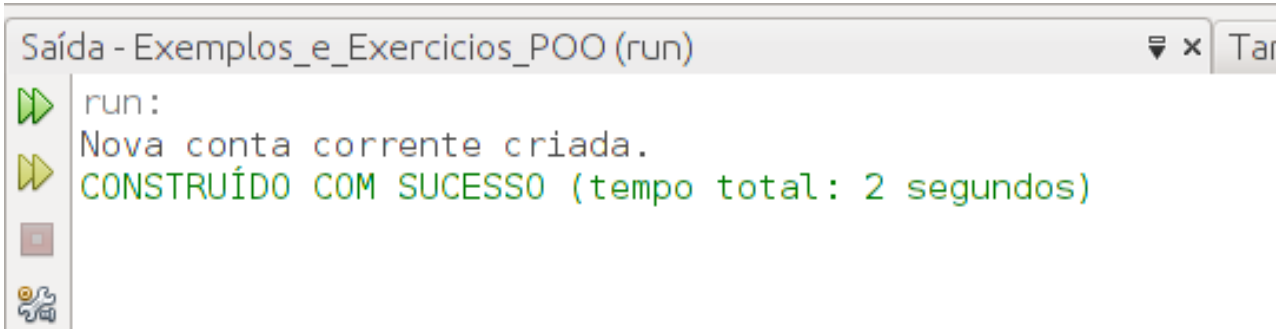
- Com o construtor definido, todo objeto criado irá executar os comandos que se encontram no corpo do método construtor.

- No caso do exemplo anterior, a partir de agora, ao instanciar um novo objeto do tipo ContaCorrente, a mensagem “Nova conta corrente criada” será exibida, pois esse é o comando que está definido no construtor da classe ContaCorrente.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Construtores

```
public class TestarContaCorrente{
    public static void main(String [] args)

        ContaCorrente conta = new ContaCorrente();
    }
}
```



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Construtores

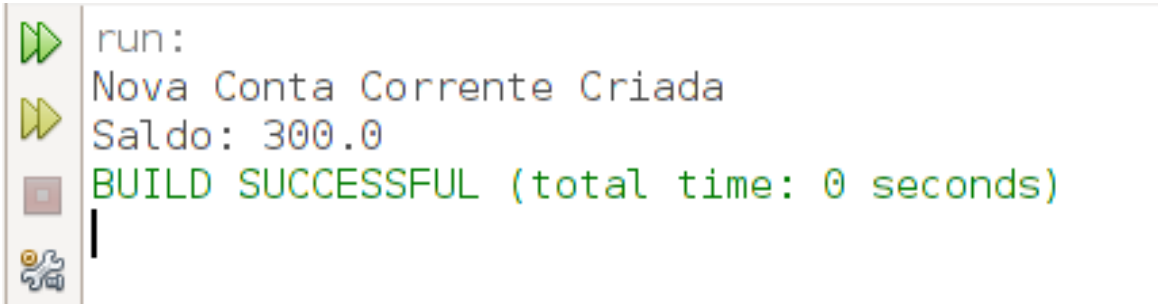
- É possível definir parâmetros nos construtores podendo assim inicializar algum tipo de informação:

```
public class ContaCorrente{
    double saldo;

    //Consturtor da classe
    ContaCorrente(double saldoInicial) {
        System.out.println("Nova conta criada");
        saldo = saldoInicial;
    }

}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Construtores

```
public class TestarContaCorrente{
    public static void main(String [] args)

        ContaCorrente conta = new ContaCorrente(300);
        System.out.println("Saldo: " + conta.saldo);
    }
}
```



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Por que usar construtores?

- Por que os construtores são úteis ou necessários?
  - Eles dão possibilidades ou obrigam o usuário de uma classe de passar argumentos para o objeto durante o processo de criação
  - No exemplo anterior, ao criar uma conta corrente,o valor do saldo inicial deve, necessariamente ser informado.
  - Não é possível criar a conta se esse valor não for informado

A palavra reservada this

- ***this*** pode ser usado para diferenciar um atributo do objeto de um parâmetro do método:

```
ContaCorrente(float saldo) {
   this.saldo = saldo;
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo

```
public class Automovel{
    String cor;
    String modelo;
    int velocidade;

    Automovel(String cor, String modelo, int velocidade) {
        this.cor = cor;
        this.modelo = modelo;
        this.velocidade = velocidade;
    }
    ...
}
```

```
public class TestarAutomovel {
    public static void main(String [] args){
        Automovel a1 = new Automovel("Azul", "Fiat", 100);     }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Sobrecarga de construtores

- É possível criar mais de um construtor em uma mesma classe, entretanto, eles devem possuir assinaturas diferentes(quantidade ou tipos de parâmetros diferentes):

```
public class ContaCorrente{
    float saldo;

    ContaCorrente() {
        System.out.println("Nova conta criada");
    }

    ContaCorrente(float saldo) {
        this.saldo = saldo;
    }
}
```

Sobrecarga de Construtores

- Quando for criar um objeto, é possível escolher qual construtor utilizar, de acordo com a necessidade:

```
public class TestarContaCorrente{
    public static void main(String [] args){
        ContaCorrente c1 = new ContaCorrente();
        ContaCorrente c2 = new ContaCorrente(100);
    }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

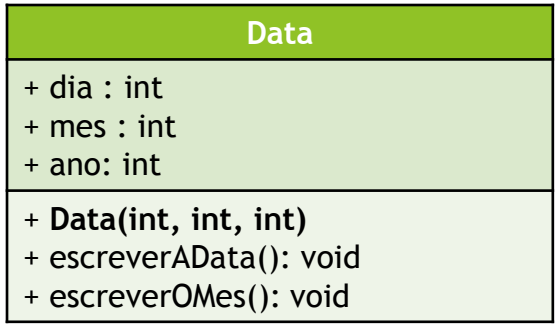
Sobrecarga de Métodos

- O mesmo princípio de sobrecarga vale para os demais métodos. Ex:

```
public class Calculadora{
      int x;
      int y;

      int somar (){
            return x + y;
      }

      float somar (float a, float b){
            return a + b;
      }

      int somar (int a, int b, int c){
            return a + b + c;
      }

      void somar(int a, int b){
            int r = a + b;
            System.out.println("A soma é: " + r);
      }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício



- Crie a classe data, conforme diagrama de UML ao lado.
- Crie um construtor para a classe Data que receba por parâmetro três valores inteiros referentes ao dia, mês e ano e atribua os valores passados por parâmetro aos atributos dia, mês e ano, respectivamente.
- Crie uma nova classe, TestarData, para testar a classe criada. Nessa classe, crie um método main que realize as seguintes operações:
  - Crie um objeto do tipo Data com o nome hoje, utilizando o construtor criado na questão a. Passe por parâmetro o dia, o mês e o ano correspondente à data de hoje.
  - Chame o método escreverAData do objeto hoje para mostrar a data na tela
  - Crie um objeto do tipo Data com o nome natal e passe por parâmetro os valores correspondentes ao dia do Natal (exemplo: dia 25, mês 12, ano 2012)
  - Chame o método escreverAData do objeto natal para mostrar a data na tela
- Execute a classe TestarData.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Encapsulamento

- Serve para ocultar os dados
- Evita que o usuário acesse membros que ele não precisa manipular ou manipule-os de forma incorreta
- Proteção do código
- Permite a modificação interna de uma classe sem alterar a sua funcionalidade e o modo como é acessada
- Utilizado em Java através dos modificadores de acesso.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Proteção do código

```
public class Data{
      int dia;
      int mes;
      int ano;

      ...
}
```

```
public class TestarData{
      public static void main(String [] args){
            Data hoje = new Data();
            hoje.dia = 32;
            hoje.mes = 13;
            hoje.ano = 2001;
      }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Proteção do código

- Os tipos de modificadores existentes são:
  - public
  - protected
  - pakage
  - private

Proteção de código

- Adicione o modificador de acesso **private** aos atributos da classe data. Em seguida, tente modificar o valor dos atributos

```
public class Data{
      private int dia;
      private int mes;
      private int ano;

      ...
}
```

```
public class TestarData{
      public static void main(String [] args){
            Data hoje = new Data();
            hoje.dia = 32;
            hoje.mes = 13;
            hoje.ano = 2001;
      }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Modificadores de acesso

- **public**: é visível em qualquer lugar
- **protected**: só é visível na mesma classe e em suas subclasses
- ***package***: default. Só é visível em classes do mesmo pacote
- **private**: só é visível dentro da mesma classe.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Modificadores de acesso

- private é um modificador de acesso (também chamado de modificador de visibilidade)
- Marcando um atributo como privado, fechamos o acesso a ele a partir de outras classes.
- É uma prática quase que obrigatória proteger os atributos de suas classes como private
- Há também o modificador public, que permite a todos acessarem um determinado atributo ou método
- É muito comum que atributos sejam private e quase todos os métodos sejam públic (não é uma regra)
  - Assim, toda conversa de um objeto com outro é feita através de troca de mensagem (acessando seus métodos)

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Modificadores de acesso

- Para a visibilidade package não existe um nome padrão para o modificador.
- Se o atributo/método não possuir um modificador declarado, sua visibilidade será package
  - Até o momento, todos os nossos atributos e métodos possuíam a visibilidade package

Notação UML

- Os modificadores de acesso possuem uma notação específica na UML.

Modificador	Representação
public	+
private	-
protected	#
package	~

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Notação UML

- Exemplo:
  - Classe Data com atributos private e métodos public

Data
- dia : int - mes : int - ano: int
+ Data(int, int, int) + escreverAData(): void + escreverOMes(): void

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Métodos getters e setters

- Para permitir o acesso aos atributos que são declarados como private de uma maneira controlada, a prática mais comum é criar dois métodos
  - Um que retorna o valor do atributo
  - E outro que altera o valor do atributo
- O padrão para esses métodos é colocar a palavra get ou set antes do nome do atributo

*OBS: O padrão do método **get** não vale para variáveis do tipo **boolean***
- *Esses atributos são acessados via **is** e **set***
- *Exemplo: para verificar se uma lâmpada está acesa, seriam criados os métodos **isLigado** e **setLigado***

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Métodos getters e setters

```
public void setNOME_DO_ATRIBUTO(<TIPO_DO_ATRIBUTO> <NOME_DO_ATRIBUTO>){
      this.<NOME_DO_ATRIBUTO> = <NOME_DO_ATRIBUTO>;
}
```

```
public void setDia(int dia){
      this.dia = dia;
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Métodos getters e setters

```
public <TIPO_DO_ATRIBUTO> get<NOME_DO_ATRIBUTO>(){
     return <NOME_DO_ATRIBUTO>;
}
```

```
public int getDia(){
     return dia;
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo

```
public class Data{
      int dia;
      int mes;
      int ano;

      public void setDia(int dia){
            this.dia = dia;
      }
      public void setMes(int mes){
            this.mes = mes;
      }
      public void setAno(int ano){
            this.ano = ano;
      }
      public int getDia(){
            return dia;
      }
      public int getMes(int mes){
            return mes;
      }
      public int getAno(int ano){
            return ano;
      }
}
```

Métodos getters e setters

- Mas o código ainda não impede que qualquer valor seja atribuído aos atributos da classe.
- Nos casos em que se faz necessário, pode-se incluir condições aos métodos set. Ex:

```
public class Data{

     ...

     public void setDia(int dia){
          if ( (dia > 0 ) && (dia <= 31){
               this.dia = dia;
          }
     }

     ...
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Métodos getters e setters

- Quando os atributos possuem algum tipo de restrição, é importante que em todos os lugares do código em que se altere os valores dos atributos, os métodos **set** sejam usados. Mesmo na própria classe:

```
public class Data{
     ...
     //construtor
     public Data(int dia, int mes, int ano){
          setDia(dia);
          setMes(mes);
          setAno(ano)

     }
     ...
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

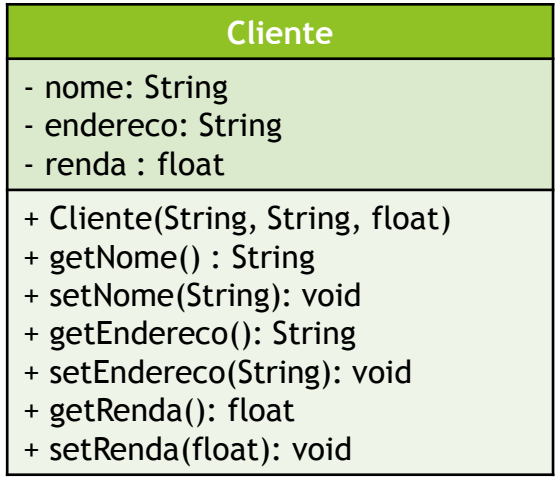
Métodos getters e setters

```
public class TestarData{
     public static void main(String [] args){
          Data hoje = new Data(18, 01, 2013);
          System.out.println("Dia: " + hoje.getDia());
          hoje.setDia(19);
          System.out.println("Dia: " + hoje.getDia());
          hoje.setDia(35);
          System.out.println("Dia: " + hoje.getDia());
     }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

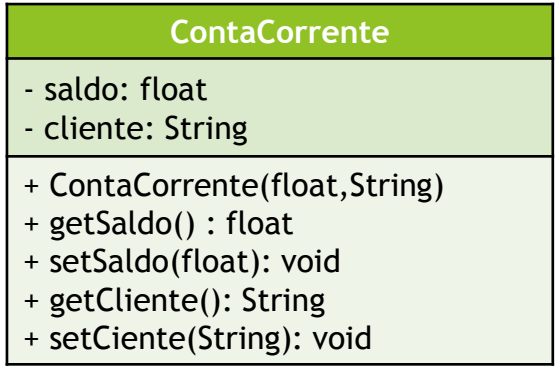
- Crie uma classe cliente que obedeça à descrição da representação UML abaixo. Perceba que você deve criar o construtor da classe e os métodos get e set para cada um dos atributos.
  - Crie a classe TestarCliente e instancie 3 objetos do tipo Cliente. Defina o valor que desejar para os atributos (nome, endereço e renda)



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercício

- Altere a classe ContaCorrente para que o atributo saldo passe a ser do tipo private. Inclua um atributo cliente do tipo String. Crie os métodos get e set para os 2 atributos. No método setSaldo, não permita que um valor negativo seja atribuído ao saldo. Caso o valor seja negativo, defina 0 (zero) como sendo o novo valor. Crie também um construtor para a classe. Siga a descrição UML abaixo. Mantenha os métodos anteriores (não representados na descrição abaixo):



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Referências

- SIERRA, Katy; BATES, Bert. Use a cabeça JAVA. Ed 2, Editora Altabooks;
- SIERRA, Katy; BATES, Bert. SCJP – Certificação Sun para Programador Java. Editora Editora Altabooks;
- Material do ESJUG – Grupo de Usuários Java do Espírito Santo – Programação Orientada Objetos. Março de 2008.
- LIGUORI, Robert; LIGUORI, Patricia. Java Guia de bolso. Editora Alta Books.
- Material produzido pela empresa Argonavis – Helder da Rocha. Programação Orientada Objetos.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br