Curso de Extensão

Programação de Aplicações Comerciais em JAVA

Aula 4

A Linguagem JAVA

("Episódio V: O Retorno de Jedi")

Prof. Monael

Método Construtores

- É um método executado automaticamente quando o objeto é criado
  Sintaxe:

<MESMO NOME DA CLASSE>(<argumentos>){ }

Exemplo:

```
class Exemplo
{
    Exemplo(int arg1)
    {           }
}
```

- Possui o mesmo nome da Classe
- Não possui valor de retorno
- Não retornam nenhum tipo de valor

Laboratório

- Vendo um método construtor em ação: Escreva o seguinte programa exemplo:

```
class Exemplo
{
    Exemplo()
    {
        System.out.println("Estou Nascendo!!!");
    }
    public static void main(String[] arguments)
    {
        Exemplo ex = new Exemplo();
    }
}
```

Laboratório 2

- Vendo um método construtor em ação:
  Escreva o seguinte programa exemplo:

```
class Exemplo2
{
    Exemplo2(String nome)
    {
        System.out.println("Oi papai. Estou nascendo! Me chamo " + nome + " Jr.");
    }
    public static void main(String[] arguments)
    {
        Exemplo2 ex = new Exemplo2("<SEU NOME>");
    }
}
```

Laboratório 3

- Na Classe Conta definida na aula anterior, defina um método construtor, onde sejam passados como argumento de criação o Nome e CPF. Atribua esses valores aos atributos pertinentes da classe Conta, gere um número aleatório para a conta e imprima um extrato inicial.

Sobrecarga de Métodos

- Em Java é possível ter vários métodos com o mesmo nome. O que diferencia estes métodos são:
  - O número de argumentos que recebem;
  - O tipo de dado ou objeto que cada um recebe.

  Exemplo:

```
class Exemplo
{
    void teste()  {                          }
    void teste(int arg1)   {                 }
    void teste(String arg1)          {       }
    void teste(int arg1, String arg2){       }
}
```

- Possui o mesmo nome da Classe
- Não possui valor de retorno
- Não retornam nenhum tipo de valor

Laboratório

- Para fixar o conceito de sobrecarga de método escreva o seguinte programa:

```
class Exemplo 3{
    void teste()            {       System.out.println( "passei no 1");}
    void teste(String a)    {        System.out.println( "passei no 2"); }
    public static void main(String[] arguments){
            Exemplo3 ex = new Exemplo3();
            ex.teste();
            ex.teste("oi");
}  }
```

Laboratório 2

- Na classe Conta defina um outro método deposito que tenha como argumento o valor do deposito e o percentual de imposto que será pago sobre o depósito. Calcule a dedução do imposto e efetue o depósito.
  Execute o programa primeiro depositando R$ 100,00 de forma normal (o primeiro método deposito) em seguida deposite mais R$ 100,00 com imposto de 10%. Observe o saldo!

Sobrecarga de Construtor

- Da mesma forma que é possível sobrecarregar métodos, os construtores também podem ser sobrecarregados:

  Exemplo:

```
class Exemplo
{
    Exemplo()                       {                   }
    Exemplo(int arg1)               {                   }
    Exemplo(String arg1)            {       }
    Exemplo(int arg1, String arg2){         }
}
```

- Possui o mesmo nome da Classe
- Não possui valor de retorno
- Não retornam nenhum tipo de valor

Laboratório

- Defina outro construtor para a classe Conta, onde um saldo inicial para a conta é aceita.
  Teste o programa executando uma instância com os dados: (nome, CPF) e outra instância com (nome, CPF e Saldo Inicial).

Modificadores

- Os modificadores que controlam o acesso a uma classe, método ou atributo são:
  - public, protected e private
- Modificadores para criar métodos e atributos de classe:
  - static
- Modificador que finaliza as classes, métodos e atributos:
  - final
- Modificador para criar métodos e classes abstratas:
  - abstract
- Modificadores usados para threads
  - synchronized e volatile

Modificadores

- Sintaxe:

  <modificador> class <Nome da Classe>

  <modificador> <tipo de retorno> <nome do metodo>(<argumentos>)

  <modificador> <tipo> <nome do atributo> [= valor;]

- Exemplos:
  - public class Conta{
  - public void saque(int valor){}
  - private float saldo;
  - static final PI = 3.141618f;
  - public abstract class Conta{

Controle de Acesso (Encapsulamento)

- Acesso Privado – (private)
  - O método ou atributo só pode ser acessado da própria classe. Qualquer outra classe não tem acesso.
  - Sintaxe:

    private <tipo> <nome do atributo>;

    Private <tipo de retorno> <nome do método>(<argumentos>){}
  - Exemplo:

    private float saldo;

    private void deposito(float valor){ }

Laboratório

- Para fixar o funcionamento do controle de acesso privado, implemente o seguinte programa:

```
class Exemplo4{
    private int numero;
    private void mostraNumero()
    {
        System.out.println(numero);
    }
    public static void main(String[] arguments)
    {
        Exemplo4 ex = new Exemplo4();
        ex.numero = 7;
        ex.mostraNumero();
    }    }
```

Controle de Acesso (Encapsulamento)

- Acesso Público – (public)
  - O método ou atributo pode ser acessado de qualquer outra classe.
  - Sintaxe:

    public <tipo> <nome do atributo>;

    public <tipo de retorno> <nome do método>(<argumentos>){}
  - Exemplo:

    public float saldo;

    public void deposito(float valor){ }

Laboratório

- Para fixar o funcionamento do controle de acesso público, implemente o seguinte programa:

```
class Exemplo4{
    public int numero;
    public void mostraNumero()
    {
        System.out.println(numero);
    }
    public static void main(String[] arguments)
    {
        Exemplo4 ex = new Exemplo4();
        ex.numero = 7;
        ex.mostraNumero();
    }    }
```

Controle de Acesso (Encapsulamento)

- Acesso Protegido – (protected)
  - O método ou atributo pode ser acessado de qualquer subclasse ou do mesmo pacote da classe que tem possui o método um atributo.
  - Sintaxe:

    protected <tipo> <nome do atributo>;

    protected <tipo de retorno> <nome do método>(<argumentos>){}
  - Exemplo:

    protected float saldo;

    protected void deposito(float valor){ }

Laboratório

- Para fixar o funcionamento do controle de acesso público, implemente o seguinte programa:

```
class Exemplo4{
    protected int numero;
    protected void mostraNumero()
    {
        System.out.println(numero);
    }
    public static void main(String[] arguments)
    {
        Exemplo4 ex = new Exemplo4();
        ex.numero = 7;
        ex.mostraNumero();
    }    }
```

Controle de Acesso (Encapsulamento)

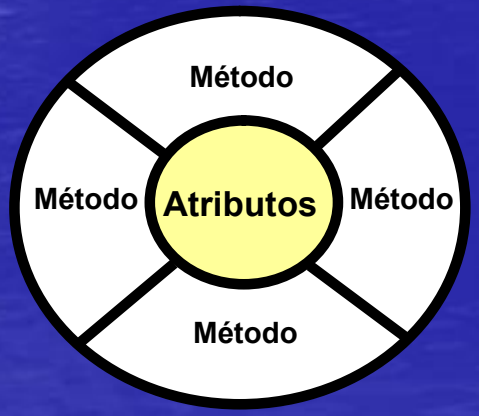
Visibilidade	public	protected	private
Da mesma classe	Sim	Sim	Sim
Qualquer classe no pacote	Sim	Sim	Não
Qualquer classe fora do pacote	Sim	Não	Não
De uma subclasse no mesmo pacote	Sim	Sim	Não
De uma subclasse fora do pacote	Sim	sim	Não

Controle de Acesso (Encapsulamento)

- Métodos Acessadores
  - Normalmente os atributos são privados ou protegidos e os métodos são públicos.

    Dessa forma para se ter acesso aos atributos usam-se os métodos acessadores.

    Assim é garantido o encapsulamento.



Controle de Acesso (Encapsulamento)

- Métodos Acessadores
  - Exemplo:

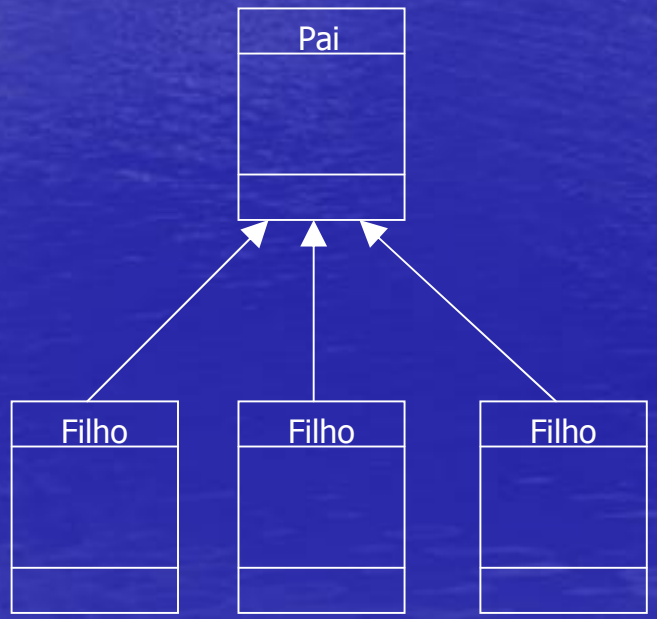
```
Public class Exemplo
{
    private float saldo;
    protected String nome;
    public void   setSaldo(float valor){   }
    public float  getSaldo()       {       }
    public void setNome(String n){         }
    public String getNome()        {       }
}
```

Laboratório

Incremente a classe Conta com modificadores de visibilidade nos atributos e métodos. E também defina métodos acessadores para todos os atributos dessa classe.

Herança

- É um mecanismo pelo qual uma classe passa a utilizar atributos e métodos definidas em uma outra classe definida como ancestral.



Herança

- Em JAVA a herança é definida pela palavra chave extends.

  Sintaxe:

  class <NOME CLASSE FILHO> extends <NOME CLASSE PAI>

  Exemplo:

  class Aluno extends Pessoa

  class Funcionario extends Pessoa

  class Humano extends Mamiferos

Laboratório

Como sabemos existem vários tipos de contas bancárias. Conta Corrente, Conta Poupança, Conta Salário, Conta Conjunta, Conta Investimento entre outras.
Anteriormente definimos a classe Conta, de uma forma genérica. Agora desejamos criar uma classe para conta poupança e uma conta investimento.
Crie uma conta poupança herdando os atributos e métodos da classe Conta. Entretanto ContaPoupanca possui um atributo adicional o aniversário, crie-o juntamente com seus métodos acessadores.

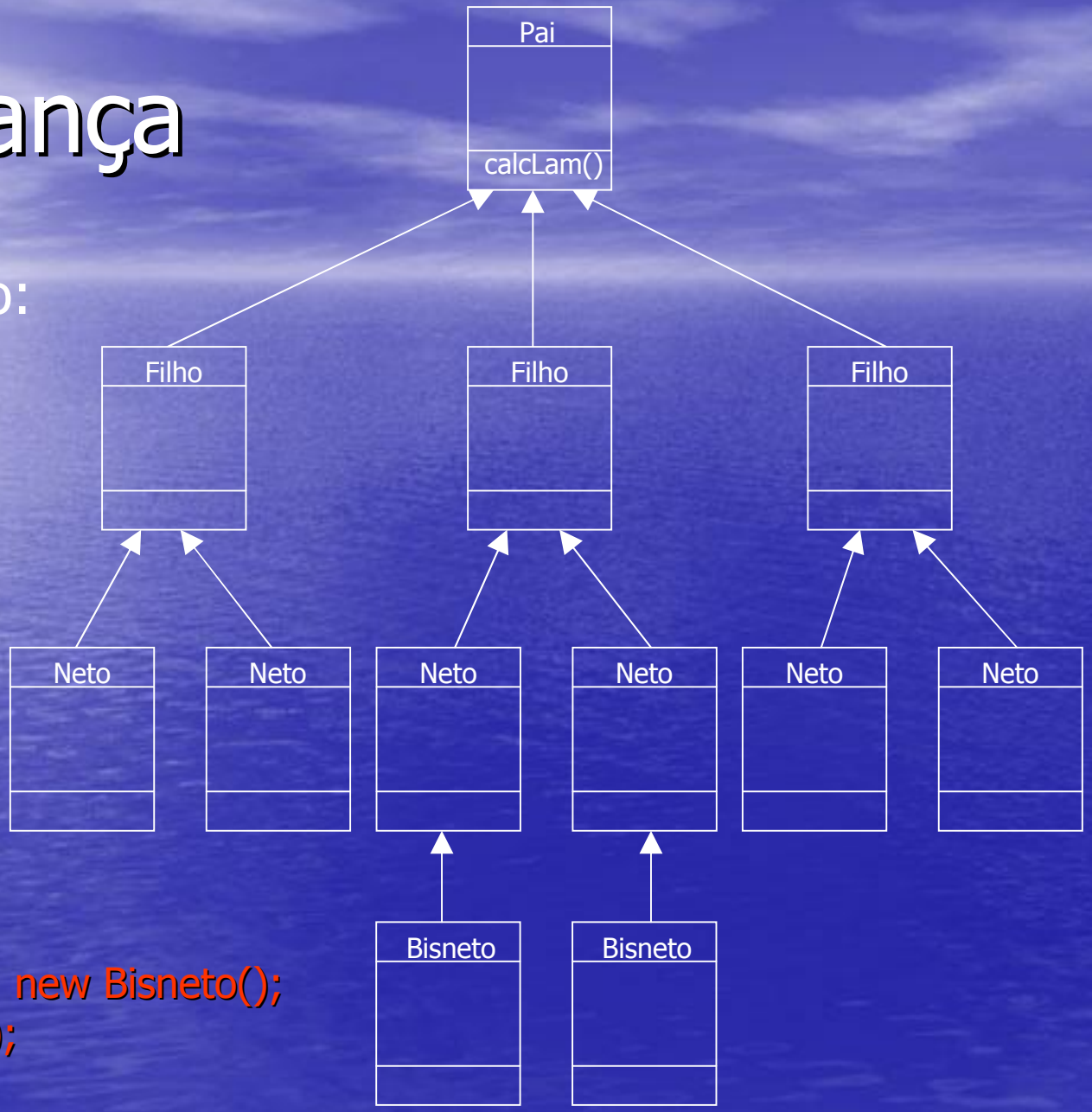
Após isso, instancie objetos da classe ContaPoupanca e manipule-os.

Herança

- Redefinindo Método
  - Quando se invoca um método do objeto, JAVA percorre a lista de métodos do objeto, caso não exista esse método JAVA sobe pela "família" de classes desse objeto procurando esse método.

Herança

Exemplo:



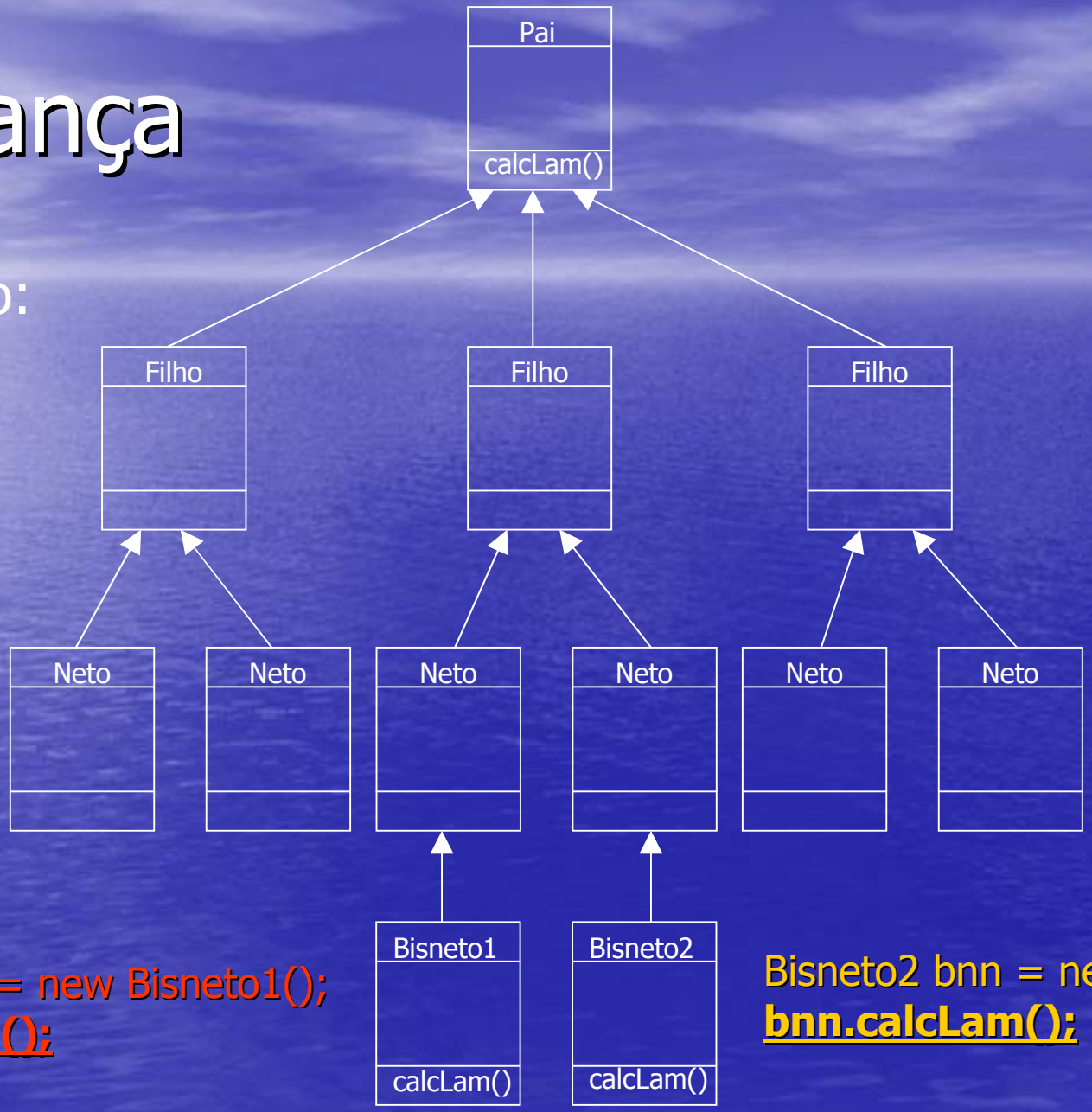

```
Bisneto bn = new Bisneto();
bn.calcLam();
```

Herança

- Redefinindo Método
  - Dessa forma em qualquer nível da herança pode-se redefinir um método das classes ancestrais.
    Dessa forma podemos em determinado ponto invocar dois métodos de forma idêntica, porém com resultados totalmente diferentes.

Herança

Exemplo:



Bisneto1 bn = new Bisneto1();
bn.calcLam();

Bisneto2 bnn = new Bisneto2();
bnn.calcLam();

Laboratório

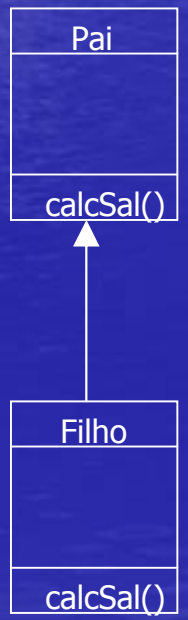
Na classe conta existe um método chamado juros() o qual foi herdado para ContaPoupanca e ContaInvestimento.

Entretanto sabemos que os juros são calculados de forma diferente para estes tipos de contas. Para conta Poupança é com base na TR e para conta Investimento depende do tipo de investimento, mas vamos supor que seja um investimento de renda fixa com base em títulos públicos prefixados pela taxa base de juros. Sendo assim, redefina o método juros para ambas as subclasses.

Herança

- Chamando método original
  - Caso seja necessário chamar o método original, presente na superclasse.

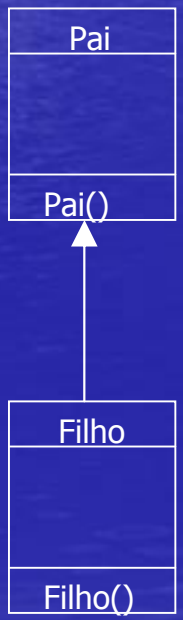
```
super.calcSal();
```



Herança

- Construtores
  - Quando uma classe filho é instanciada é executado o seu construtor. No entanto pode-se chamar o construtor da classe pai da seguinte forma:

```
super ();
```



Laboratório

- Vamos definir o construtor das classes ContaPoupanca e ContaInvestimento de uma forma mais "elegante".
- Receba todos os argumentos dê set apenas no argumento particular da subclasse e invoque o construtor da superclasse passando os argumentos necessários.
- Teste e veja se o resultado foi satisfatório.

Classes Abstratas

- As vezes é desejável orientar a implementação de uma classe.
- Em outros casos o modelo é tão amplo que uma classe acaba tornando-se muito gerais.
- Neste caso Java dispõe das classes abstratas.
- As classes abstratas:
  - Não podem ser instanciadas

Classes Abstratas

Sintaxe:

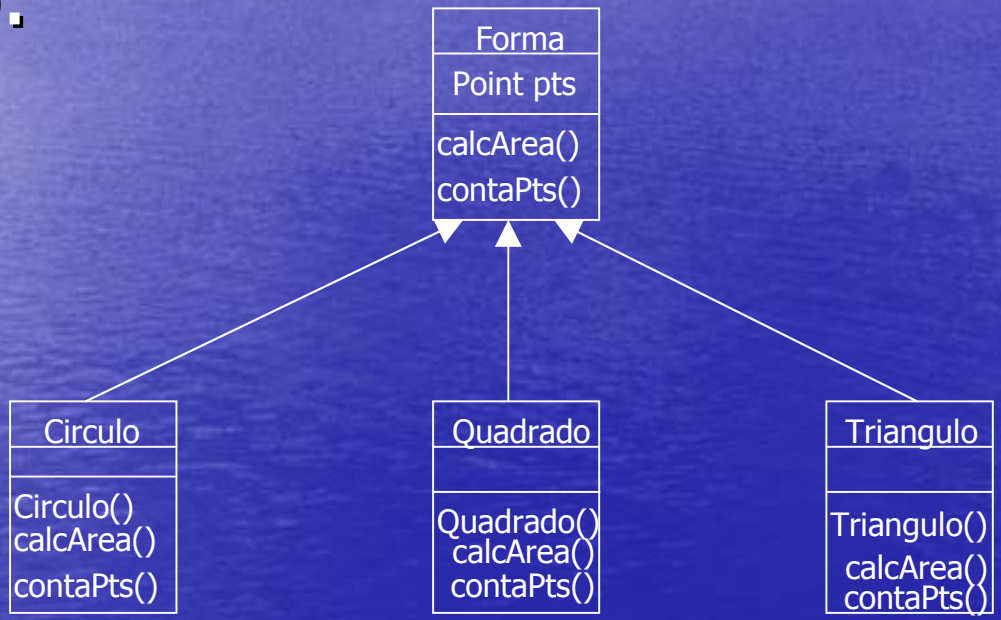
<modificador> abstract class <Nome>{}

Exemplo:

public abstract class Forma{ }

Classes Abstratas

Exemplo:



Interfaces

- Java permite se definir um tipo especial de classe chamadas Interfaces
- As Interfaces servem para promover uma espécie de herança múltipla
- Uma Interface deve seguir as regras:
  - Nenhum método declarado pode ser implementado
  - Todas os atributos devem ser estáticos e finais

Interfaces

Sintaxe:

interface <Nome da Interface>{ }

Exemplo:

interface Teste { }

Interfaces

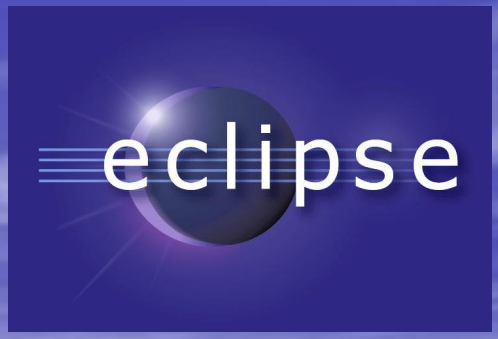
Sintaxe do uso:

class <nome> implements <interface>{}

Exemplo:

class Aluno extends Pessoas implements Direitos

Eclipse

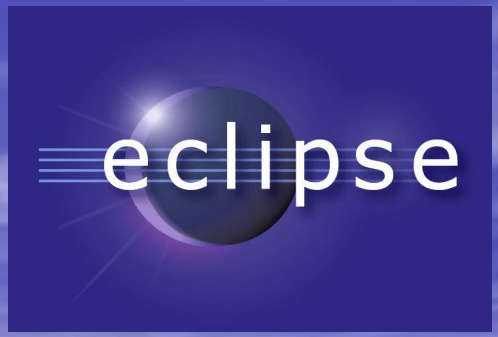


Eclipse é uma IDE aberta para a construção de programas de computador.

O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares.

Hoje o Eclipse é a IDE Java mais utilizada no mundo. Possui como características marcantes a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

Eclipse

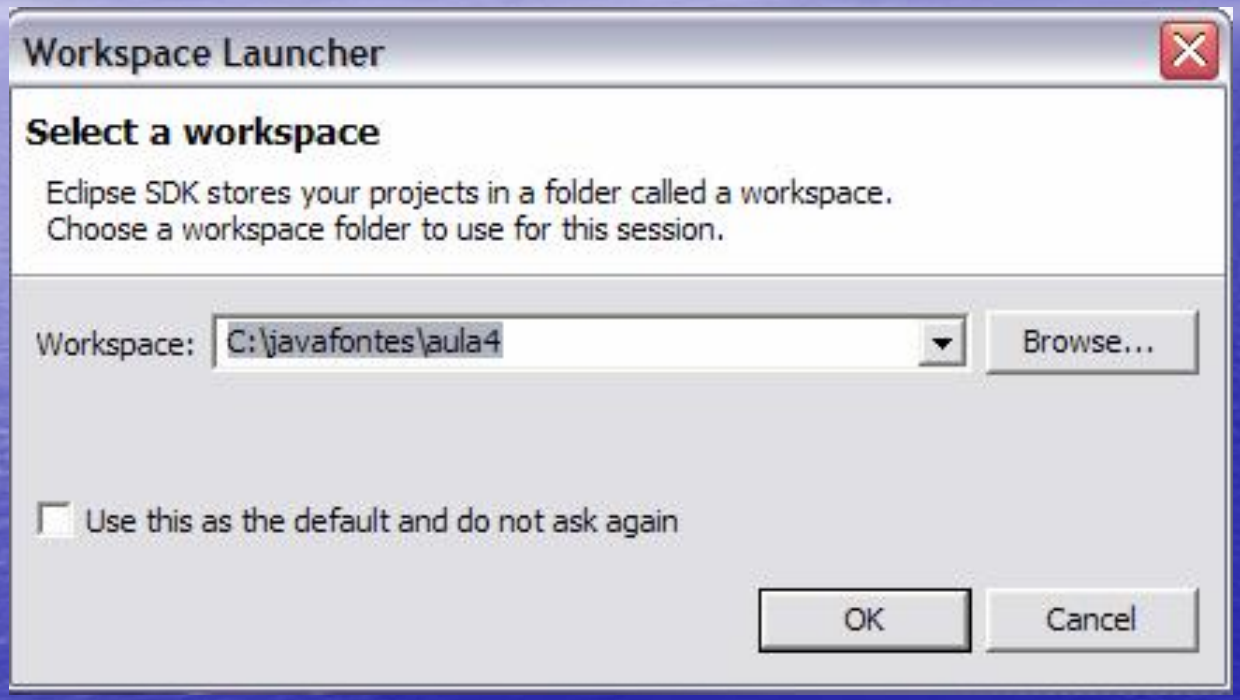


O IDE Eclipse pode ser baixado gratuitamente em:

http://www.eclipse.org/downloads/

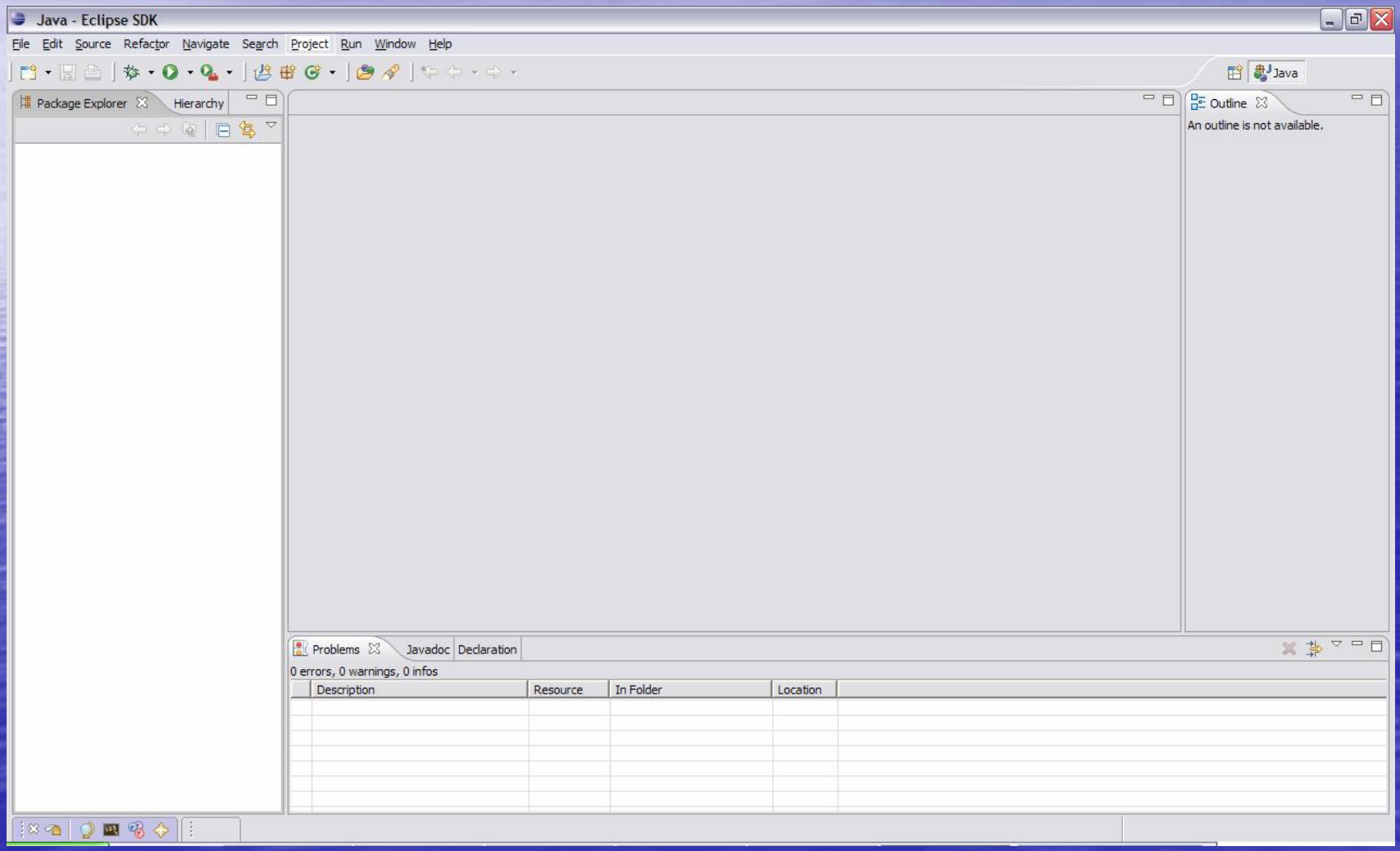
eclipse-SDK-3.2-win32.zip provavelmente é o arquivo que será baixado. Basta descompacta-lo e executar Eclipse.exe

Eclipse



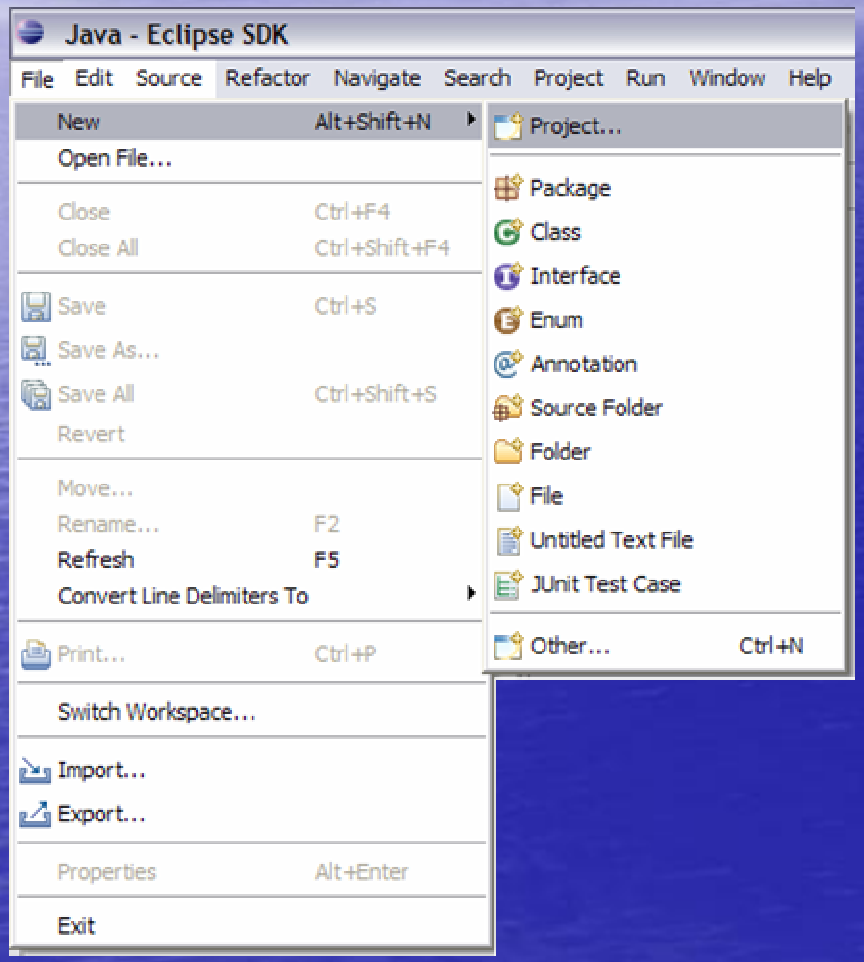
Selecione a pasta onde estará o Workspace do eclipse

Eclipse


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
Hierarchy
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 errors, 0 warnings, 0 infos
Description
Resource
In Folder
Location

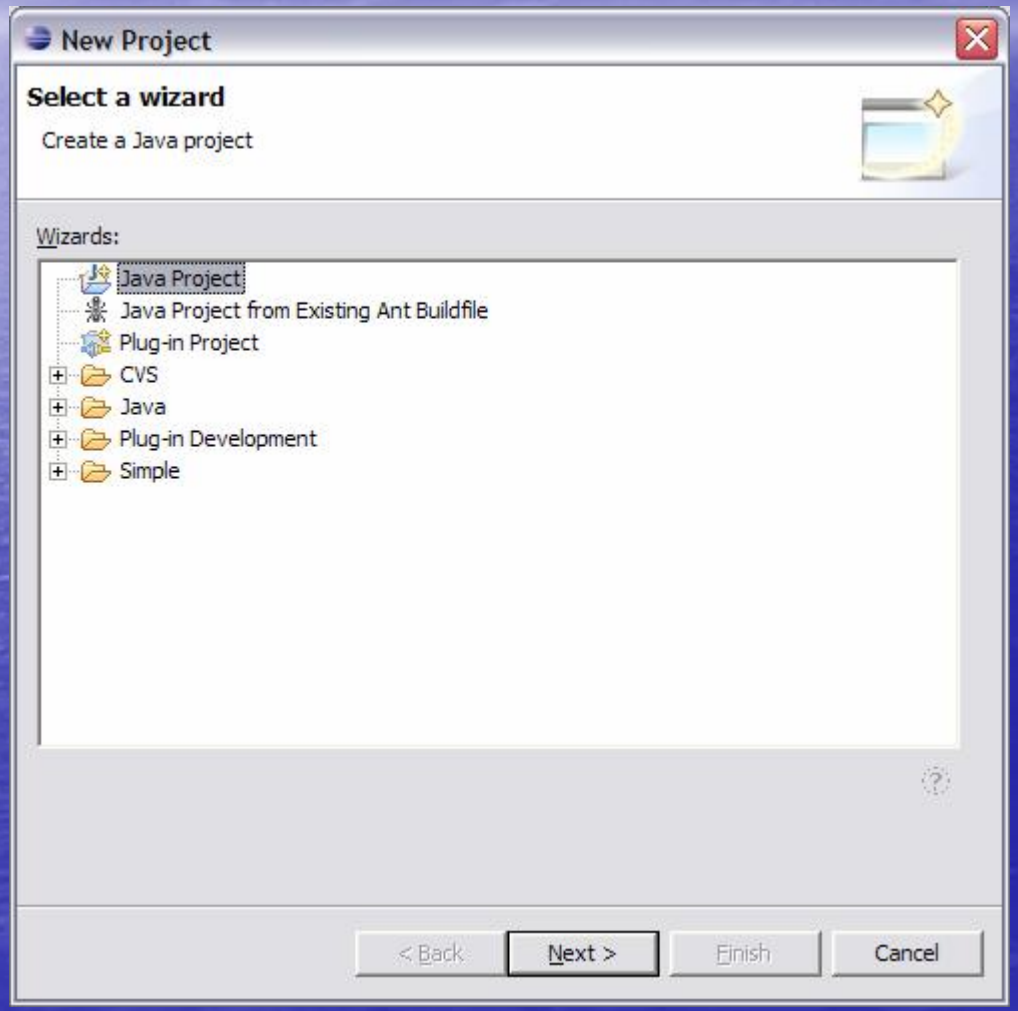
Eclipse

Criando o Projeto Conta



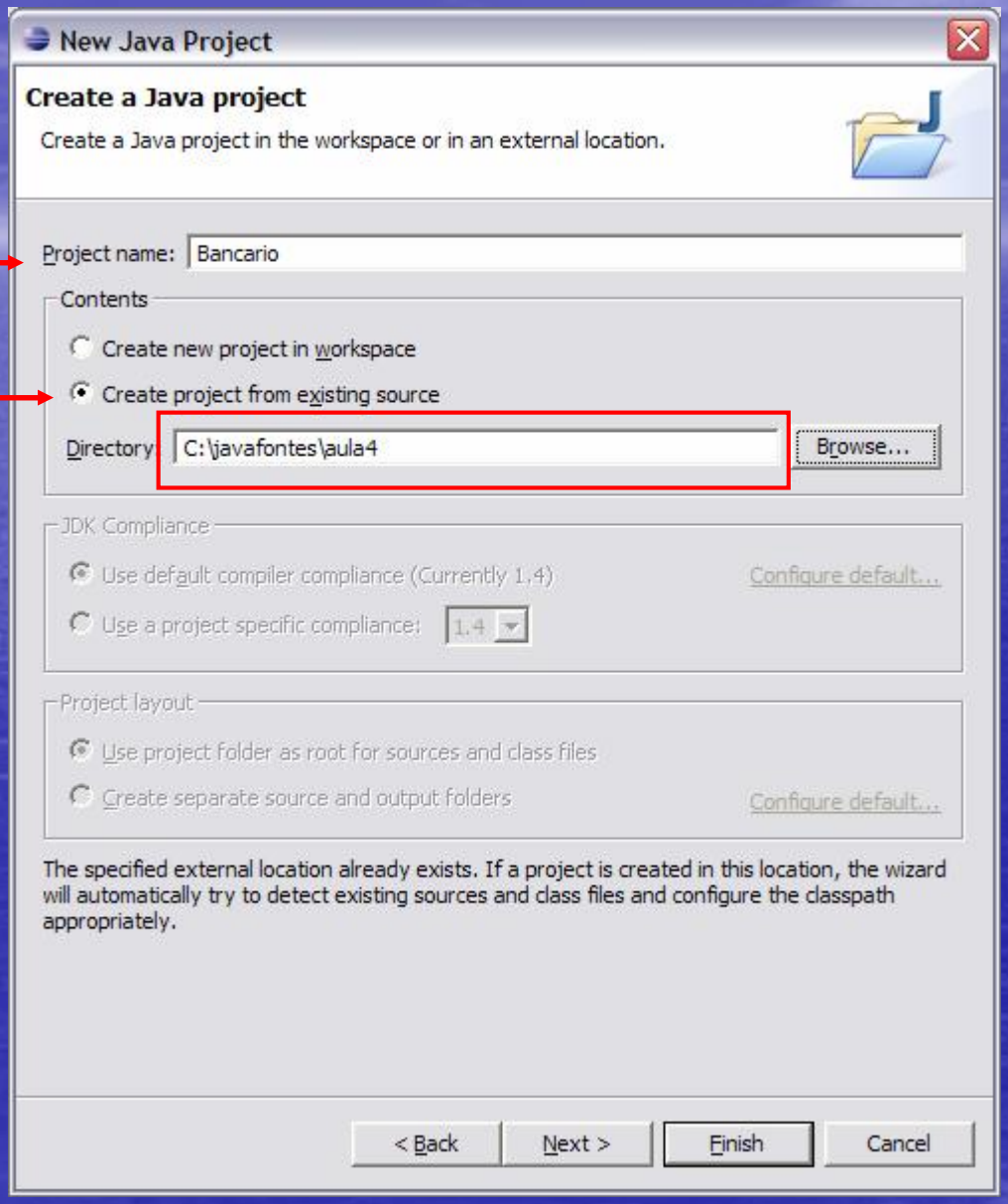
Eclipse

Criando o Projeto Conta



Eclipse

Criando o Projeto Conta



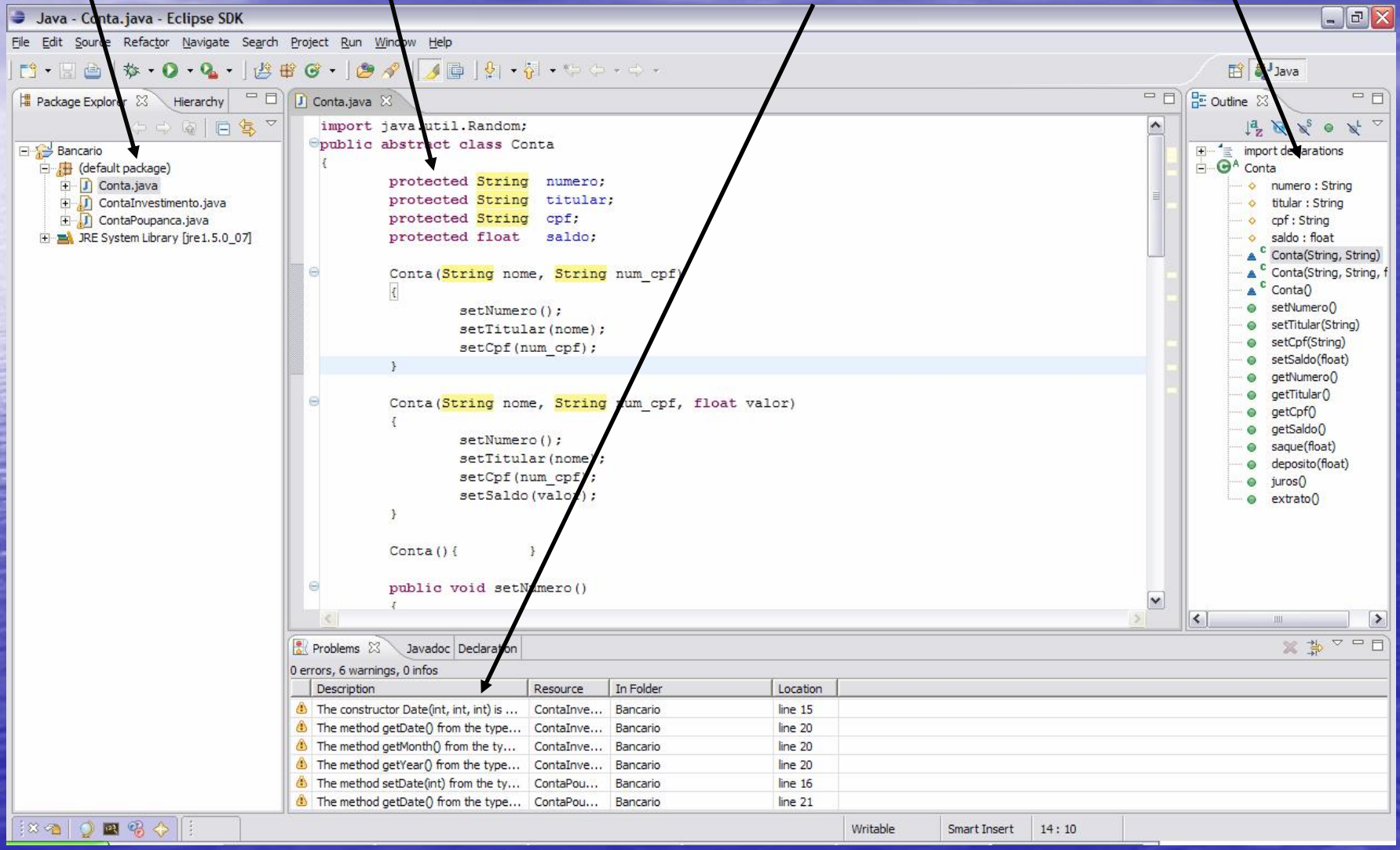
Eclipse

Projeto

Código Fonte

Resultados da (pré) compilação

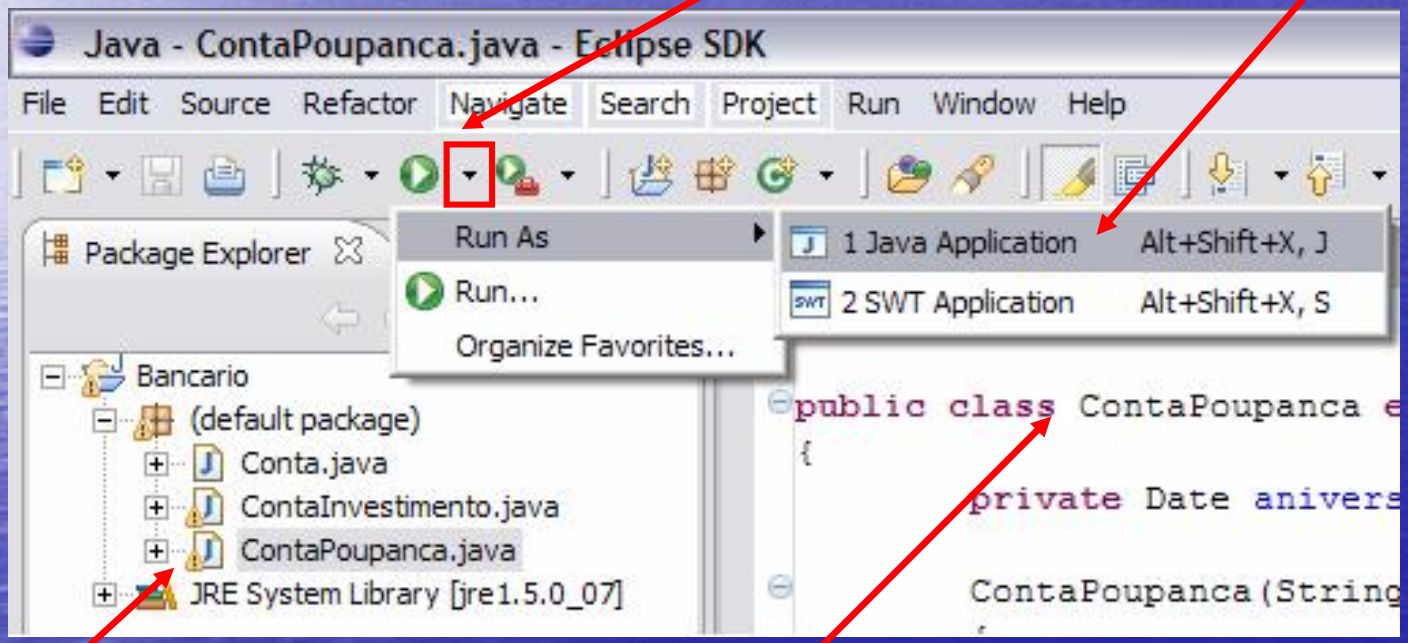
Outline da Classe



Eclipse

Executando um Programa

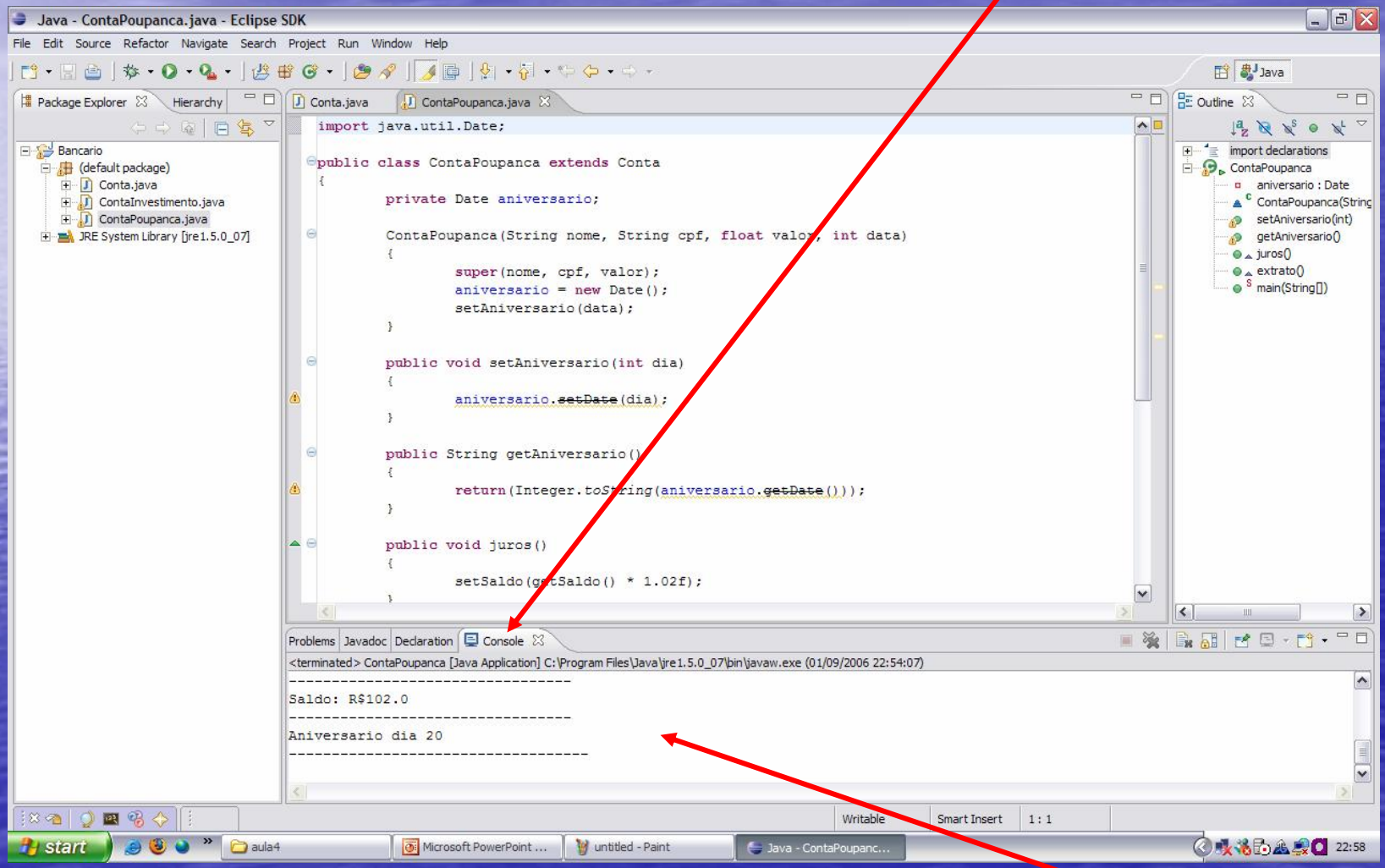
Selecione a Classe ContaPoupanca

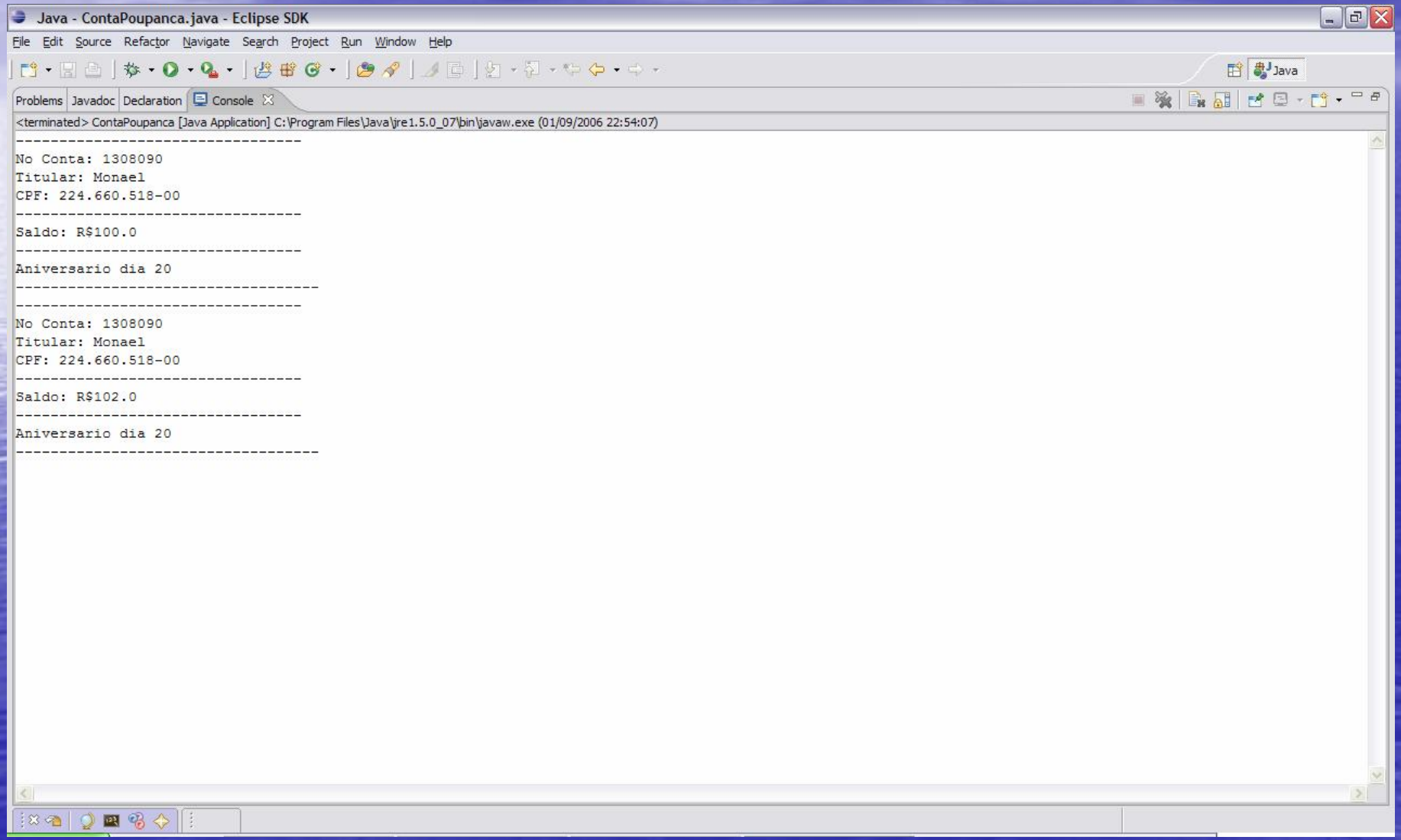


Eclipse

Vendo o Resultado

Double Click para aumentar



Eclipse

Vendo o Resultado