



# Amarração Tardia de Tipos

- Definição dinâmica do método depende da classe a ser usado: Depende do objeto que invoca o método.



# Amarração Tardia de Tipos

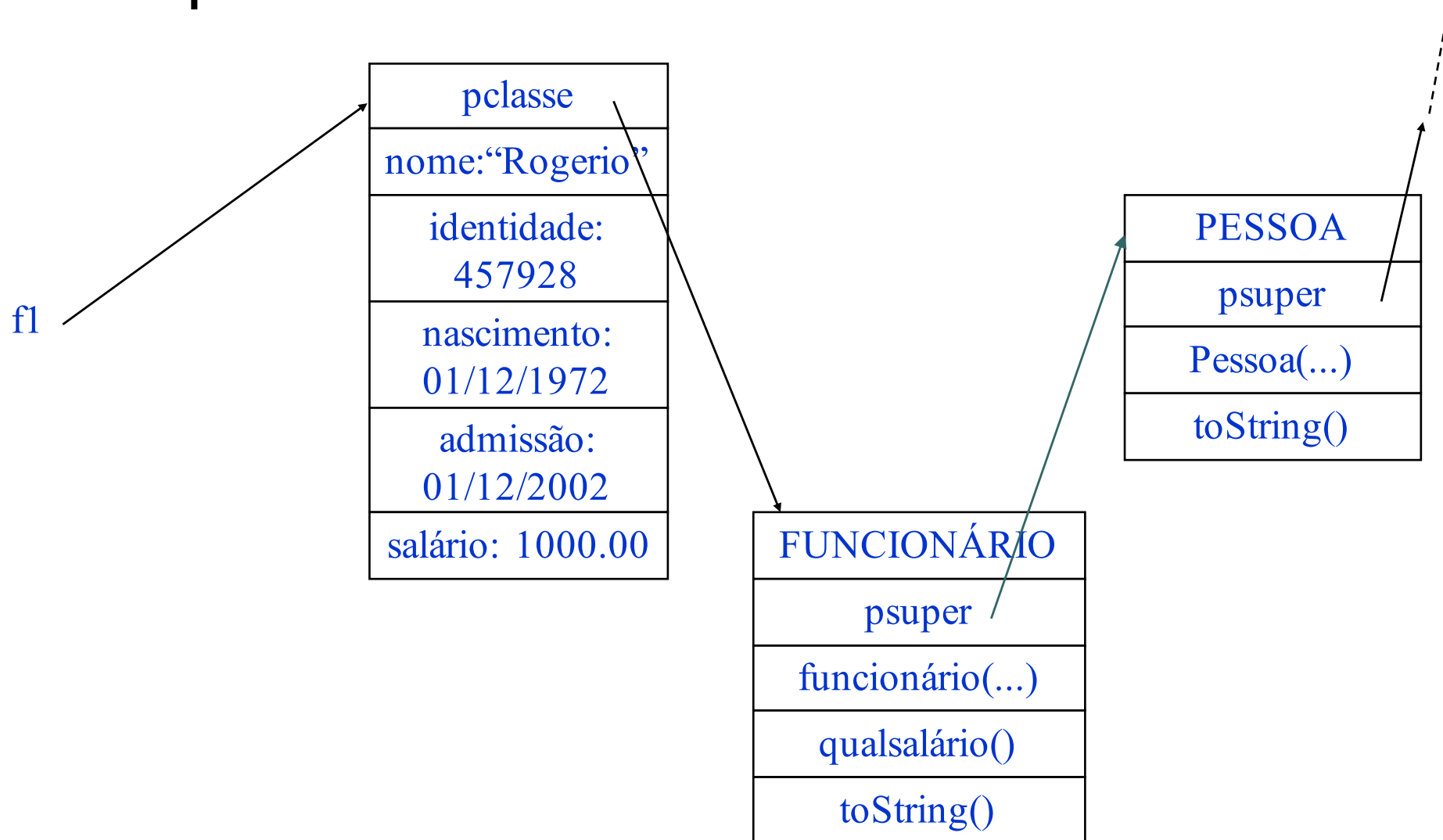
## Exemplo

```
public class Empresa {  
    public static void main(String[] args) {  
        float s; int i;  
        DataCons d1 = new DataCons((byte)12,(byte)12,(short)1967);  
        Pessoa p = new Pessoa ("Denise", 3454637, d1);  
        DataCons d2 = new DataCons((byte)1,(byte)12,(short)1972);  
        DataCons d3 = new DataCons((byte)1,(byte)12,(short)2002);  
        i = p.qualIdentidade();  
        Funcionario f1 =  
            new Funcionario ("Rogerio", 93452128 ,d2 ,d3 ,(float)1000.00);  
        s = f1.qualSalario();  
        i = f1.qualIdentidade();  
        System.out.println(f1);  
    }  
}
```

Empresa.java



# Amarração Tardia de Tipos





# Amarração Tardia de Tipos

```
public class EmpresaDin {  
    public static void main(String[] args) {  
        float s; int i;  
        DataCons d1 = new DataCons((byte)12,(byte)12,(short)1967);  
        Pessoa p = new Pessoa ("Denise", 3454637, d1);  
        DataCons d2 = new DataCons((byte)1,(byte)12,(short)1972);  
        DataCons d3 = new DataCons((byte)1,(byte)12,(short)2002);  
        Funcionario f1 =  
            new Funcionario ("Rogerio", 457928,d2,d3,(float)1000.00)
```

```
p = f1;
```

```
System.out.println(p);
```

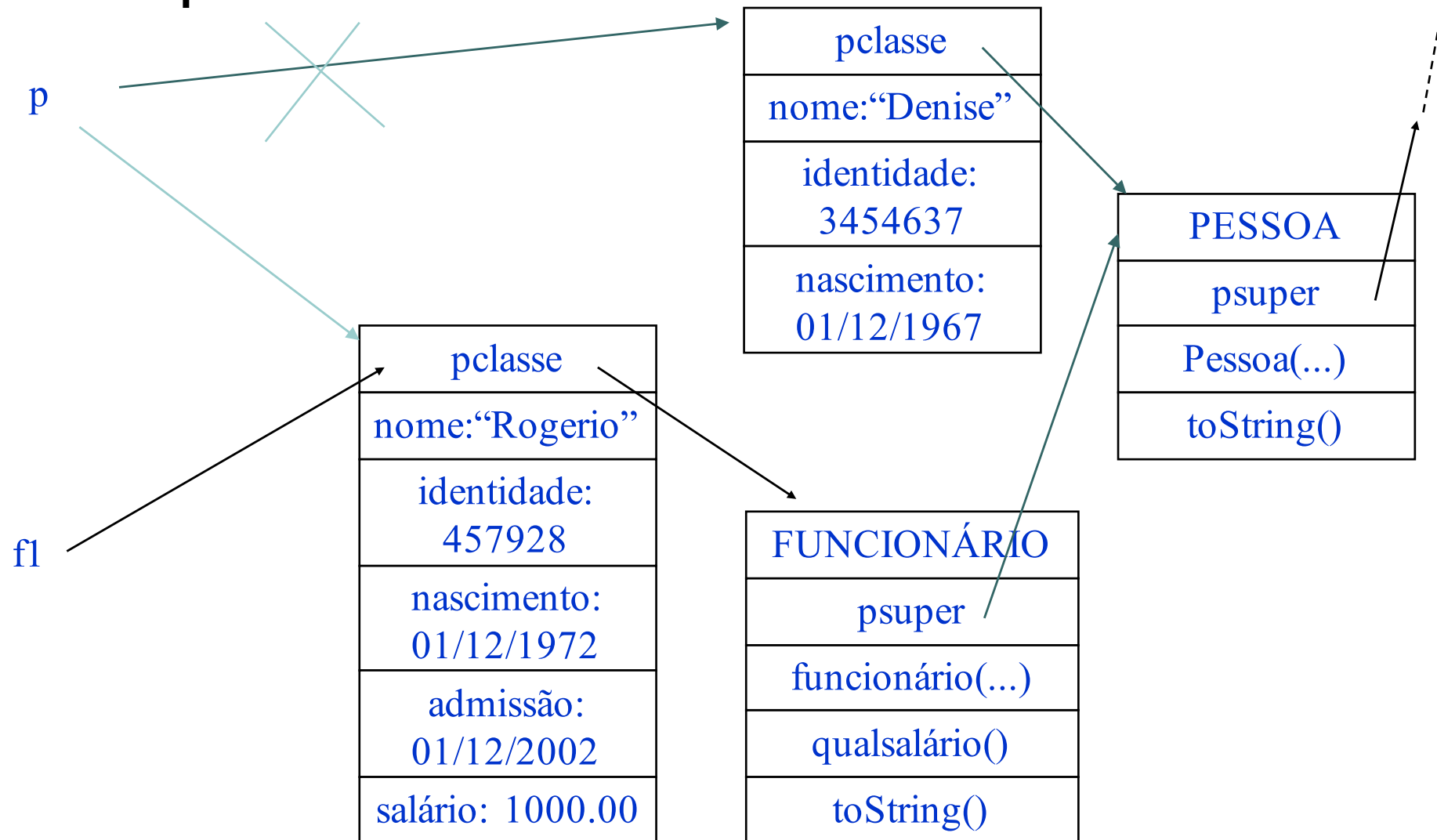
```
}  
}
```

referência de Pessoa passa a apontar  
para instância de Funcionário

invoca toString de Pessoa? de Funcionário?



# Amarração Tardia de Tipos





# Amarração Tardia de Tipos

- O Java chama isso de Ligação Dinâmica ou Dynamic Binding
- A maquina virtual sabe qual o tipo atual do objeto. Assim faz uma ligação dinâmica e quando o objeto vem de uma determinada classe, os métodos também também vem da classe.
- A maquina virtual cria uma tabela de métodos, onde para cada classe ha as assinaturas correspondentes.
- Quando executamos um exemplo como os já mostrados a JVM simplesmente acessa esta tabela.



# Polimorfismo

- A relação ***é-um-tipo-de*** entre classes permite a existência de outra característica fundamental de linguagens OO, o **polimorfismo**
- Polimorfismo (“muitas formas”)
  - Permite a manipulação de instâncias de classes que herdem de uma mesma classe ancestral de forma unificada.



# Polimorfismo

- Algumas definições
- Mecanismo que permite que uma operação receba argumentos de diferentes tipos

*Um parâmetro declarado como valor de um tipo...  
... pode receber valores do subtipo.*

- Mecanismo que permite que um método receba argumentos de diferentes classes:

*Um parâmetro declarado como instância da  
superclasse...  
... pode receber instâncias da subclasse.*





# Exemplo

```
public class ConcessionariaDeAutomoveis
{ public static void main(String args[])
{ Automovel a1 = new Automovel("Fiat","bege",
                                Automovel.movidoAAAlcool);
  AutomovelBasico a2 = new AutomovelBasico("Corsa","cinza",
                                             Automovel.movidoAGasolina);

  imprime(a1);
  imprime(a2);
}
```

O método **imprime** recebe um objeto da classe Automovel como parâmetro. Observe que nesse exemplo, chamamos o método passando a1 e a2, ou seja, objetos de classes diferentes mas da mesma hierarquia de classes, caracterizando dessa forma a utilização de polimorfismo.

```
public static void imprime(Automovel a)
{ System.out.println("Dados do automóvel escolhido : ");
  System.out.println(a.toString());
  System.out.println("Valor: "+a.quantoCusta());
  System.out.println(a.quantasPrestacoes()+" prestações de "+
                     (a.quantoCusta()/a.quantasPrestacoes()));
}
}
```



# Polimorfismo

## ○ **Vantagens do uso do Polimorfismo:**

- Permite o envio de mensagens a um objeto sem a determinação exata do seu tipo;
- Permite a extensibilidade do sistema com pouco ou nenhum impacto nas classes existentes;
- Permite a construção de classes genéricas para efetuar tarefas gerais comuns aos softwares, como as estruturas de dados.



# Polimorfismo

- “instanceof”
  - “variável referência” instanceof “nome da classe”
    - Retorna true or false



# Polimorfismo

- Uso de instanceof

- Exemplo

- Para a hierarquia Pessoa, Funcionário e Chefe de Departamento
    - Classe Empréstimo Bancário
      - A classe EmpréstimoBancario calcula o valor de empréstimos que podem ser dados a pessoas (instâncias das classes Pessoa, Funcionario e ChefeDeDepartamento) por um banco popular. Empréstimos a pessoas que não são funcionárias são de valor fixo, enquanto empréstimos dados a funcionários e chefes de departamento são baseados nos salários que estes recebem.
      - \* Esta classe demonstra polimorfismo de classes usando instâncias das classes herdeiras da classe Pessoa, e sobrecarga, implementando o mesmo método mais especializado para instâncias de classes herdeiras da classe Funcionario.

```

class EmprestimoBancario
{
public static void main(String[] argumentos)
{
// Criamos uma inst ncia da classe Pessoa
Pessoa p1 = new Pessoa("Kurt G del",10973213,
    new Data((byte)23,(byte)12,(short)1904));
// Criamos tr s inst ncias da classe Funcionario
Funcionario f1 = new Funcionario("Henri Poincar ",19283712,
    new Data((byte)12,(byte)7,(short)1897),
    new Data((byte)28,(byte)1,(short)1918),
    2500.0f);
Funcionario f2 = new Funcionario("Paul Dirac",98736812,
    new Data((byte)20,(byte)1,(short)1885),
    new Data((byte)31,(byte)3,(short)1909),
    3200.0f);
Funcionario f3 = new Funcionario("Wolfgang Pauli",33886620,
    new Data((byte)14,(byte)9,(short)1902),
    new Data((byte)16,(byte)11,(short)1930),
    3600.0f);
// Criamos uma instancia da classe ChefeDeDepartamento
ChefeDeDepartamento c1 = new ChefeDeDepartamento("Edwin Hubble",4259782,
    new Data((byte)20,(byte)1,(short)1875),
    new Data((byte)20,(byte)7,(short)1899),
    4100.0f,
    "Laboratorio de Astrofisica",
    new Data((byte)20,(byte)7,(short)1899));
// Calculamos os empr stimos que podem ser feitos a cada pessoa
System.out.println(calculaEmprestimo(p1));
System.out.println(calculaEmprestimo(f1));
System.out.println(calculaEmprestimo(f2));
}
}

```

```
public static float calculaEmprestimo(Pessoa p)
{
    return 1000.0f; // qualquer pessoa pode ter um emprestimo de 1000 reais.
} // fim do metodo calculaEmprestimo
```

```
public static float calculaEmprestimo(Funcionario f)
{
    float emprestimo = 0f; // inicialmente consideramos o emprestimo zero
    // Primeiro verificamos se a instancia é uma instancia da classe
    // ChefeDeDepartamento. Se for, calculamos o emprestimo como sendo quatro
    //vezes o salario recebido.

    if (f instanceof ChefeDeDepartamento)
    {
        emprestimo = 4.0f*f.qualSalario();
    }
    // Se a instancia nao for da classe ChefeDeDepartamento, verificaremos se ela
    // é instancia da classe Funcionario e, se for, calculamos o emprestimo como //sendo duas
    //vezes o salario recebido.
    else if (f instanceof Funcionario)
    {
        emprestimo = 2.0f*f.qualSalario();
    }
    return emprestimo;
} // fim do metodo calculaEmprestimo

} // fim da classe EmprestimoBancario
```



# Polimorfismo

- Refazer a classe EmprestimoBancário com o conceito de conversão de tipos de classe
- Pesquisar e trazer a solução para apresentação na próxima aula (05/11)



# Conversão de Tipos entre Classes

- Cast
- Assim como tipos primitivos
  - Instâncias de Classes podem ser convertidas para outras
    - Implícita
    - Explícita





# Conversão Implícita

- Quando fazemos um "casting" de dados primitivos da direita para esquerda, eles ocorrem automaticamente:

byte -> short -> int -> long -> float -> double

- Isso ocorre, pq quem está na direita é "mais genérico" do que quem está na esquerda.
  - Ou seja, vc faz um "upcasting". O contrário seria um *downcasting*.

Exemplo:

```
int i = 5;  
long j = i;    //Temos um upcasting ou conversao implícita.  
byte b1 = i;  //Errado. "Type Mismatch".  
byte b2 = (byte) i ; //Correto. Downcasting ou conversão  
                    explicito é necessário.
```



# Conversão Implícita

- Upcasting
- A conversão deve ser da classe mais específica para a mais genérica.
- As classes devem estar em uma hierarquia
  - Descendente -> Ancestral
  - Subclasse -> SuperClasse
    - Parent p = new Child();
    - p=c
      - P referencia da classe pai, c referencia da classe filha
  - Já vimos isto no slide 41
  - Blue J, exercício Empresa, Funcionário, Gerente.



# Conversão de Tipos entre Classes

- Conversão explícita das instâncias de classes
  - DownCasting
    - (nome da classe filha) variável referência da classe pai
      - Exemplo
        - `ChefeDepartamento aux = (ChefeDepartamento)p`
        - P é um variável para instâncias da Classe Pessoa.



# Upcast X DownCast

Pessoa

nome, identidade, nascimento

Pessoa(n,i,nasc);

toString();

Funcionário

admissão, salário

Funcionário(n,i,nasc,adm,sal);

qualSalário();

toString();

ChefeDeDepartamento

departamento, promoçãoAChefe

ChefeDeDepartamento(n,i,nasc,adm,sal,dep,prom);

qualDepartamento();

toString();

DownCast

Upcast



## Exemplo

- Se na Classe EmprestimoBancario (slide 51) quiséssemos fazer um 'so método para calcular o valor do empréstimo para Pessoa, Funcionário e Chefe de Departamento?

```

public static float calculaEmprestimo(Pessoa p)
{
    float emprestimo = 1000.f;
    if (p instanceof ChefeDeDepartamento)
    {
        ChefeDeDepartamento temporario = (ChefeDeDepartamento)p;
        emprestimo = 4.0f*temporario.qualSalario();
    }
    else if (p instanceof Funcionario)
    {
        Funcionario temporario = (Funcionario)p;
        emprestimo = 2.0f*temporario.qualSalario();
    }
    return emprestimo;
} // fim do metodo calculaEmprestimo

} // fim da classe EmprestimoBancarioComCast

```

Não podemos acessar o método qualSalário da instância p pois esta é uma instância da classe Pessoa. Devemos criar uma instância temporária da classe ChefeDeDepartamento a partir de p e usa-la para chamar o método qualSalário. Notem que não é necessário inicializar a instância com a palavra-chave new, ela será somente outra referência a p.