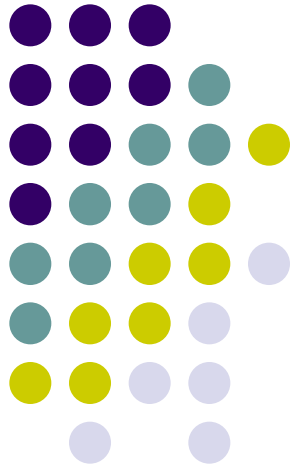
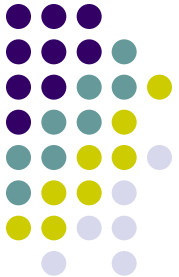
COM220
Aula 10: Polimorfismo

Prof. Laércio Baldochi

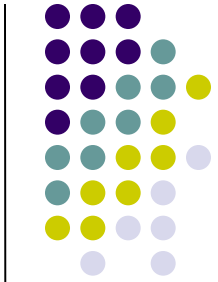


Conteúdo



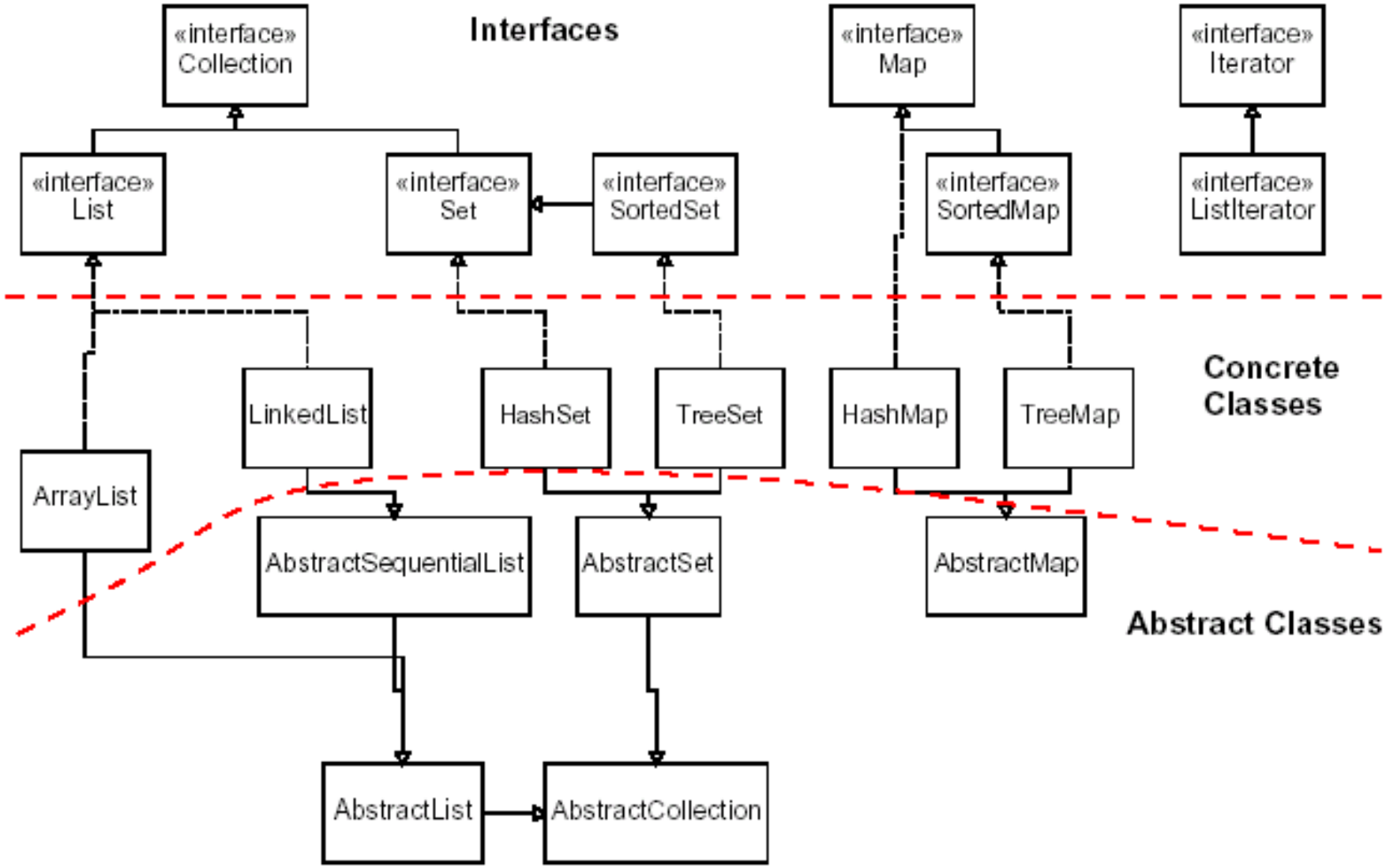
- Coleções
- Polimorfismo
- Classes abstratas
- Métodos abstratos

Coleções

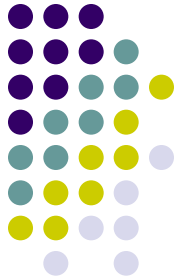


- Coleção
  - Um objeto que armazena dados
    - Estrutura de dados
  - Algumas coleções permitem ordenar os dados armazenados
  - Operações típicas
    - add, remove, clear, contains(chave), size, ...
  - Classes de coleção estão no pacote java.util
    - import java.util.*;

Framework de coleções



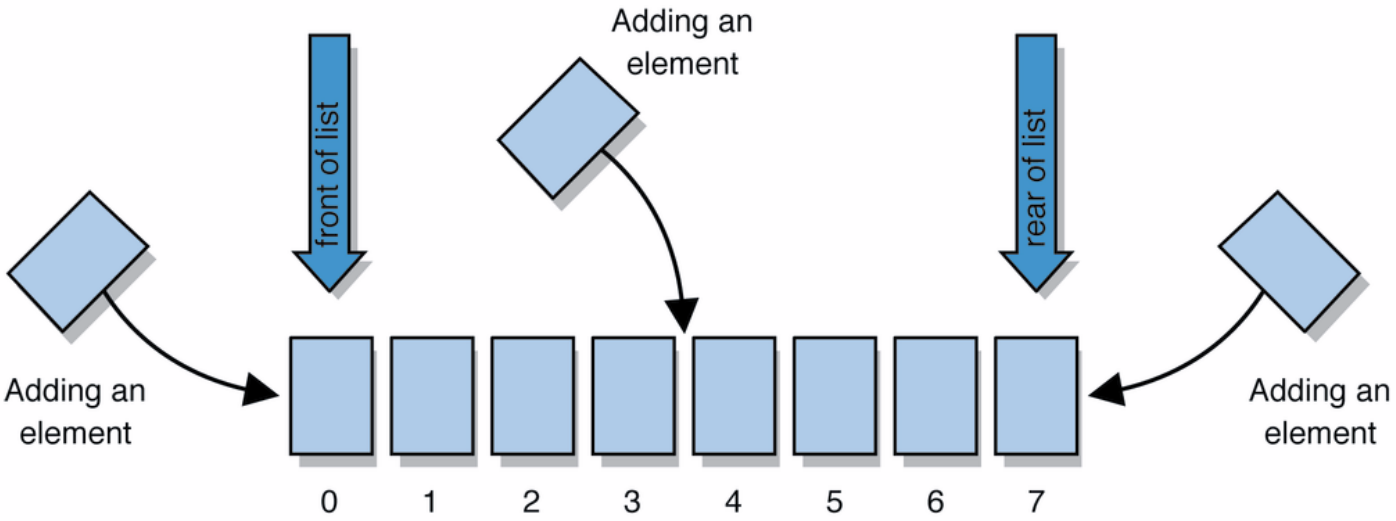
Listas



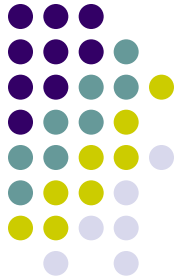
- Lista: coleção que armazena uma sequência de elementos ordenados
  - Cada elemento é acessível por meio de um indíce
  - Tem um tamanho (número de elementos adicionados)
  - Elementos podem ser adicionados em qualquer posição
  - Em Java, uma lista é representada por meio da classe **ArrayList**

Listas

- Lista: coleção que armazena uma sequência de elementos ordenados
  - Elementos podem ser adicionados em qualquer posição



Listas



- Ideia: Criar um objeto que representa uma lista de itens (inicialmente vazia)

  []
- Pode-se inserir elementos a uma lista. A inserção padrão ocorre no final da lista

  [Olá, ABC, okay, 12345]
- O objeto gerencia os elementos que são inseridos, suas ordens, índices e tamanho da lista

  Assim, um ArrayList é um Array automaticamente redimensionável

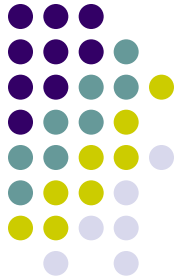
Métodos da classe ArrayList

`add(`**obj**`)`	Acrescenta **obj** no final da lista
`add(`**pos**`, `**obj**`)`	Insere **obj** na posição **pos**, deslocando os valores subsequentes à direita
`clear()`	Remove todos os elementos da lista
`indexOf(`**obj**`)`	Retorna a primeira posição onde **obj** for localizado (-1 se obj não estiver no ArrayList)
`get(`**pos**`)`	Retorna o **obj** da posição **pos** especificada
`remove(`**pos**`)`	Remove e retorna o **obj** da posição **pos**, deslocando os elementos subsequentes à esquerda
`set(`**pos**`, `**obj**`)`	Substitui **obj** da posição **pos** com novo **obj**
`size()`	Retorna o nro de elementos da lista
`toString()`	Retorna uma string representando a lista, p.e. "[3, 42, -7, 15]"

Métodos da classe ArrayList

`addAll(`**list**`)` `addAll(`**pos**`, `**list**`)`	Acrescenta os elementos de **list** a esta lista (no final da lista, ou os insere na posição pos fornecida)
`contains(`**obj**`)`	Retorna true se **obj** consta na lista
`containsAll(`**list**`)`	Retorna true se esta lista contém todos os elementos de **list**
`equals(`**list**`)`	Retorna true se **list** contém os mesmos **obj** desta lista
`iterator()`	Retorna um objeto usado para examinar o conteúdo da lista
`lastIndexOf(`**obj**`)`	Retorna a última posição em que **obj** é encontrado na lista (-1 se não for encontrado)
`remove(`**obj**`)`	Encontra e remove **obj** da lista
`removeAll(`**list**`)`	Remove todos os objs desta lista que estão em **list**
`retainAll(`**list**`)`	Remove desta lista todos os objs que não estão em **list**
`subList(`**de**`, `**para**`)`	Retorna a porção da lista entre os índices **de** (inclusive) e **para** (exclusive)
`toArray()`	Retorna os elementos dessa lista como um array

Parâmetros de tipo (Generics)

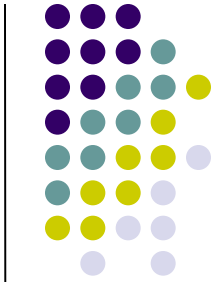


ArrayList <Type> name = new ArrayList <Type>();

Quando se constrói um ArrayList, deve-se especificar os tipos de elementos que ele deverá conter entre "<" e ">"

- Conhecido como parâmetro de tipo ou classe genérica
- Abordagem permite que a classe ArrayList armazene diferentes tipos de dados

Parâmetros de tipo (Generics)



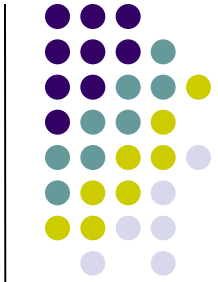
ArrayList <Type> name = new ArrayList <Type>();

Exemplo:

```
ArrayList <String> nomes = new ArrayList <String>();
nomes.add("Jose da Silva");
nomes.add("Joao de Souza");
```

Mais informações -> API

Array vs ArrayList



Construção

```
String[] nomes = new String[5];
ArrayList<String> list = new ArrayList<String>();
```

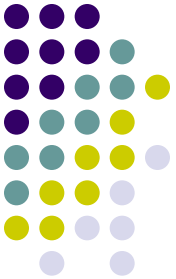
Armazenando um valor

```
nomes[0] = "Jessica";
list.add("Jessica");
```

Recuperando um valor

```
String s = nomes[0];
String s = list.get(0);
```

ArrayList vs Array

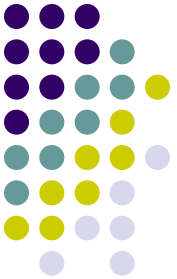


Fazendo uma operação com cada valor que começa com "B"

```
for (int i = 0; i < nomes.length; i++) {
    if (nomes[i].startsWith("B")) { ... }
}
```

```
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
}
```

ArrayList vs Array

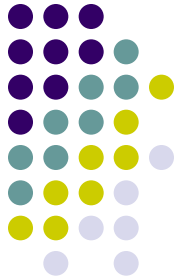


Verificando se o valor "Beto" é encontrado

```
for (int i = 0; i < nomes.length; i++) {
    if (nomes[i].equals("Beto")) { ... }
}
```

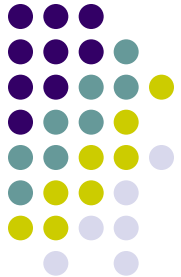
```
if (list.contains("Beto")) { ... }
```

Exercício 1



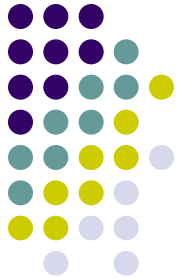
- Ler um arquivo texto e processá-lo da seguinte maneira
  - Imprimir o texto com todas as palavras no plural (que terminam em 's') escritas com letras maiúsculas.
  - Imprimir o texto invertido (de trás para a frente)

Exercício 1

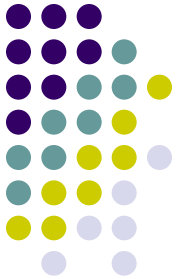


- Melhor forma de resolver o exercício consiste em representar o texto por meio de uma lista
  - Processar a lista convertendo as palavras no plural para caixa alta
  - Processar a lista de trás para frente para obter o texto invertido

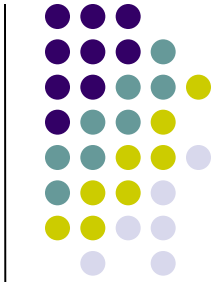
Exercício 1



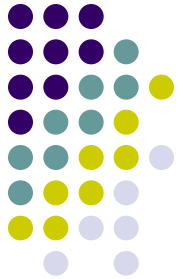
```
import java.io.*;
import java.util.*;

public class Palavras {

    ArrayList<String> palavras = new ArrayList<String>();

    public void leArq() {
        try {
            Scanner input = new Scanner(new File("texto.txt"));
            while (input.hasNext()) {
                String pal = input.next();
                palavras.add(pal);
            }
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
```



```
public void paraMaiuscula() {
    String temp = null;
    for (int i = 0; i < palavras.size(); i++) {
        if (palavras.get(i).endsWith("s")) {
            temp = palavras.get(i).toUpperCase();
            palavras.set(i, temp);

        }
    }
}
```

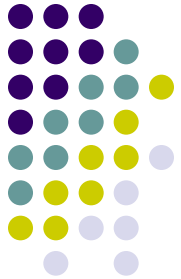


```
//public String imprimeTexto() {
//    return palavras.toString();
//}

public String imprimeTexto1(){
    String texto = "";
    for (int i = 0; i < palavras.size(); i++) {
        texto += palavras.get(i) + " ";
    }
    return texto;
}

public String imprimeTexto2(){
    String texto = "";
    for (String palavra: palavras) {
        texto += palavra + " ";
    }
    return texto;
}
```



```
    public String imprimeInverso(){
        String inverso = "";
        for (int i = palavras.size() - 1; i >= 0; i--){
            inverso += palavras.get(i) + " ";
        }
        return inverso;
    }

    public static void main(String args[]) {
        Palavras pal = new Palavras();
        pal.leArq();
        System.out.println (pal.imprimeTexto1());
        pal.paraMaiuscula();
        System.out.println (pal.imprimeTexto2());
        System.out.println (pal.imprimeInverso());
    }
}
```

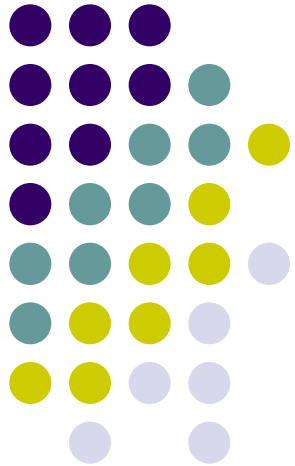
Arquivo texto.txt



1 2 3 testando. Esse é um teste para a aula de COM220.

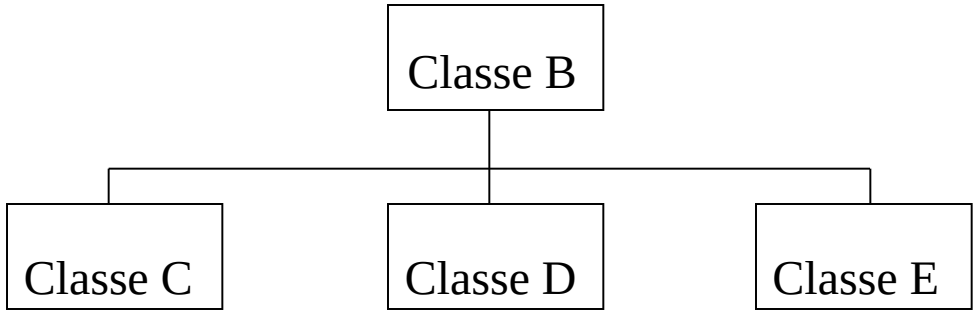
Vamos agora analisar as palavras no plural. Uma maçã, duas maçãs. Uma banana, duas bananas.

Polimorfismo

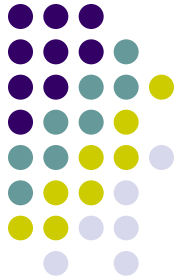


Polimorfismo

- Característica que permite implementar sistemas mais facilmente extensíveis
- Os programas podem ser escritos para processar genericamente – como objetos de superclasse – objetos de todas as classes existentes em uma hierarquia
  - Desse modo, o objeto adquire comportamento polimórfico

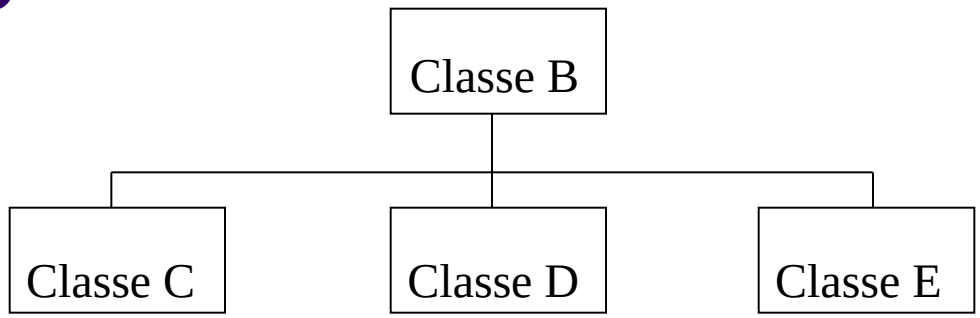


Polimorfismo



- De variáveis
  - Capacidade de assumir formas diferentes
  - Java permite a utilização de variáveis polimórficas
    - Uma mesma variável permite referência a objetos de tipos diferentes
    - Tipos permitidos são de uma determinada classe e todas as suas subclasses

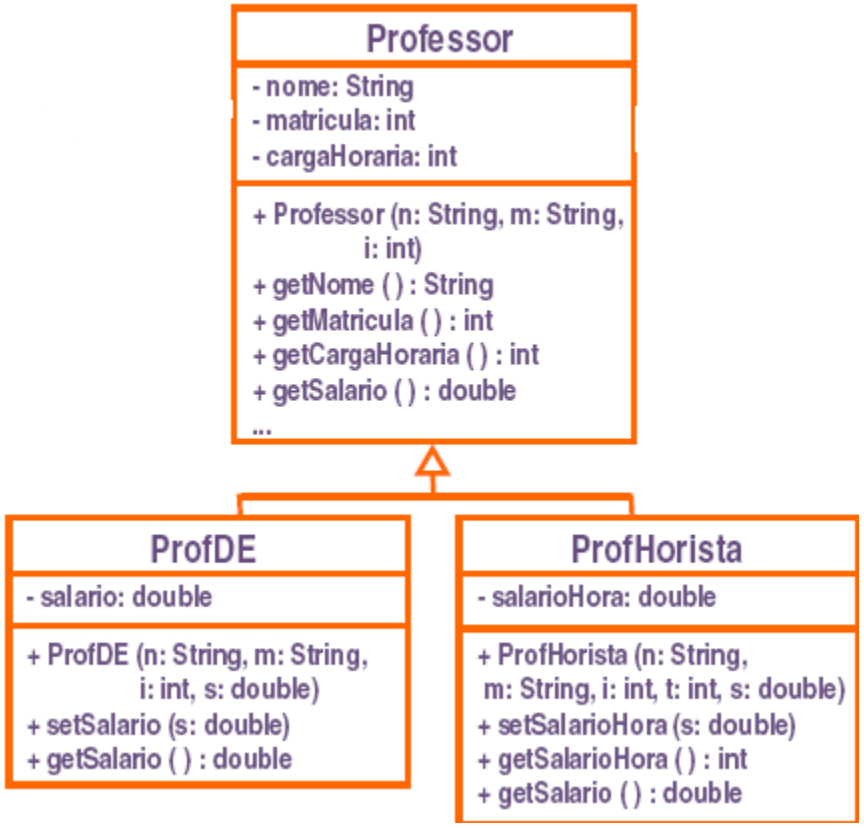
Polimorfismo



- De variáveis
  - A uma referência do tipo da superclasse pode ser atribuída uma referência da própria superclasse ou de qualquer uma de suas subclasses na hierarquia de classes
  - Exemplo
    - ClasseB cb = new ClasseB();
    - cb = new ClasseC();

Polimorfismo

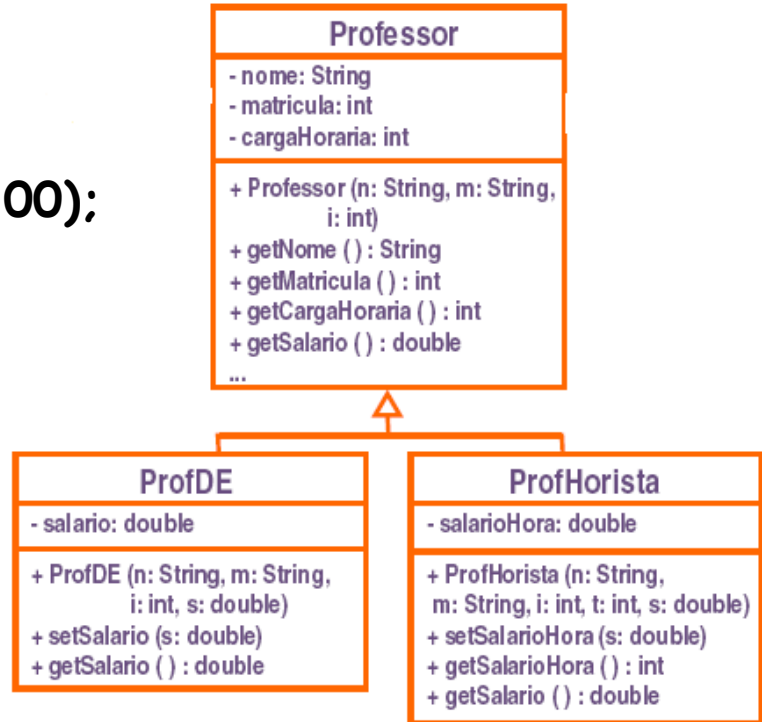
- Exemplo:
  - Classe Professor



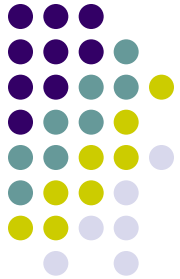
Polimorfismo

- Classe Professor

```
Professor prof1 = new ProfDE("Joao", 1, 1123.56);   \\   correto
ProfDE prof2 = new Professor ("Maria", 2, 14);   \\   erro

Professor profSuper;
ProfDE profSub;
ProfDE prof3 = new ProfDE("Jose", 3, 1500.00);
profSuper = prof3;   \\ correto
profSub = profSuper;   \\   erro
profSub = (ProfDE) profSuper;   \\ correto
```



Polimorfismo Dinâmico



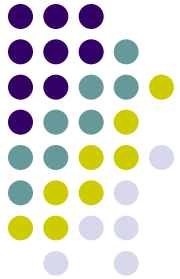
- Quando encontramos

```
Professor prof1 = new ProfDE("Joao", 1, 1123.56);
```

dizemos que *prof1* é uma referência do tipo *Professor* e o tipo do objeto armazenado em *prof1* é *ProfDE*

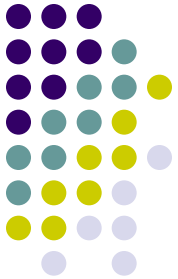
- Logo, temos dois conceitos para tipos
  - Tipo estático
    - Tipo declarado na variável (referência)
  - Tipo dinâmico
    - Tipo do objeto correntemente referenciado pela variável

Polimorfismo
Estudo de caso



- Vamos criar e manipular uma lista de professores utilizando um *ArrayList*
  - Tarefas:
    - Criar a lista
    - Inserir professores na lista
    - Visualizar o nome de todos os professores
    - Visualizar o salário/hora dos professores horistas

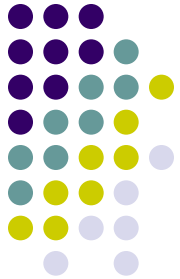
Estudo de caso
Solução 1



- **Dois ArrayList, um para cada tipo de professor**

```
...
ArrayList<ProfDE> cadDE = new ArrayList<ProfDE>();
ArrayList<ProfHorista> cadHorista = new ArrayList<ProfHorista>();
cadDE.add(new ProfDE("Joao", 1, 1123.56));
cadDE.add(new ProfDE("Maria", 2, 1200.00));
cadHorista.add(new ProfHorista("Jose", 3, 14, 12.5));
cadHorista.add(new ProfHorista("Fernando", 4, 12, 12.5));
cadHorista.add(new ProfHorista("Ana", 5, 20, 12.5));
System.out.println("Nome dos Professores DE:");
for(ProfDE p : cadDE) {
   System.out.println(p.getNome());
}
System.out.println("Salario/hora dos Professores Horistas:");
for(ProfHorista p : cadHorista) {
   System.out.println(p.getNome()+ " " + p.getSalarioHora());
}
```

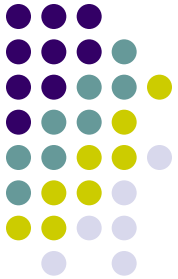
Estudo de caso
Solução 2



- **Utilizar o polimorfismo e declarar somente um ArrayList de Professor**

```
ArrayList<Professor> cadProfessor = new ArrayList<Professor>();
cadProfessor.add(new ProfDE("Joao", 1, 1123.56));
cadProfessor.add(new ProfDE("Maria", 2, 1200.00));
cadProfessor.add(new ProfHorista("Jose", 3, 14, 12.5));
cadProfessor.add(new ProfHorista("Fernando", 4, 12, 12.5));
cadProfessor.add(new ProfHorista("Ana", 5, 20, 12.5));
System.out.println("Nomes dos Professores:");
for(Professor p : cadProfessor) {
   System.out.println(p.getNome());
}
```

Estudo de caso
Solução 2

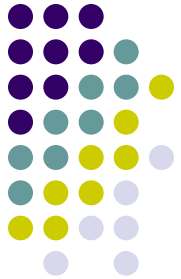


```
...
System.out.println("Salario/hora dos Professores Horistas:");
for(Professor p : cadProfessor) {
  if (p instanceof ProfHorista)
     System.out.println(p.getNome() + " " + p.getSalarioHora());
}
```

Erro de compilação, pois o método *getSalarioHora()* não pertence à classe Professor
Tipo estático X Tipo dinâmico

```
        if (p instanceof ProfHorista) {
           System.out.println(p.getNome() + " " +
                              ((ProfHorista)p).getSalarioHora());
        }
     }
  }
```

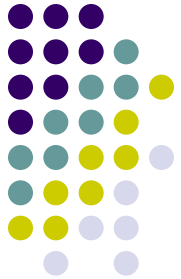
É necessário converter a referência para a classe ProfHorista (*cast*) para chamar o método



Polimorfismo

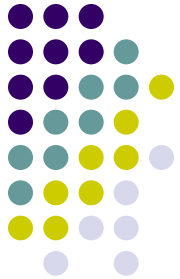
- O nosso exemplo
- Usa uma referência de um tipo único (do tipo da superclasse) para armazenar objetos variados do tipo das subclasses
- Envolve o uso automático do objeto armazenado na superclasse para selecionar um método de uma das subclasses

Polimorfismo



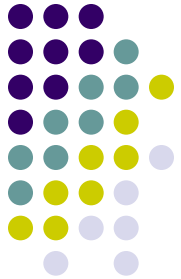
- De métodos
  - Uma *mesma* operação pode ser definida em diversas classes, cada uma implementando a operação de uma maneira própria
  - Utiliza como base a sobrescrita de métodos

Exemplo de métodos polimórficos



- Considere a classe Conta, a qual pode ser de 3 tipos diferentes (subclasses)
  - ContaCorrente
  - ContaPoupança
  - ContaSuper
- Cada tipo de conta possui o método imprimeDados

Exemplo de métodos polimórficos

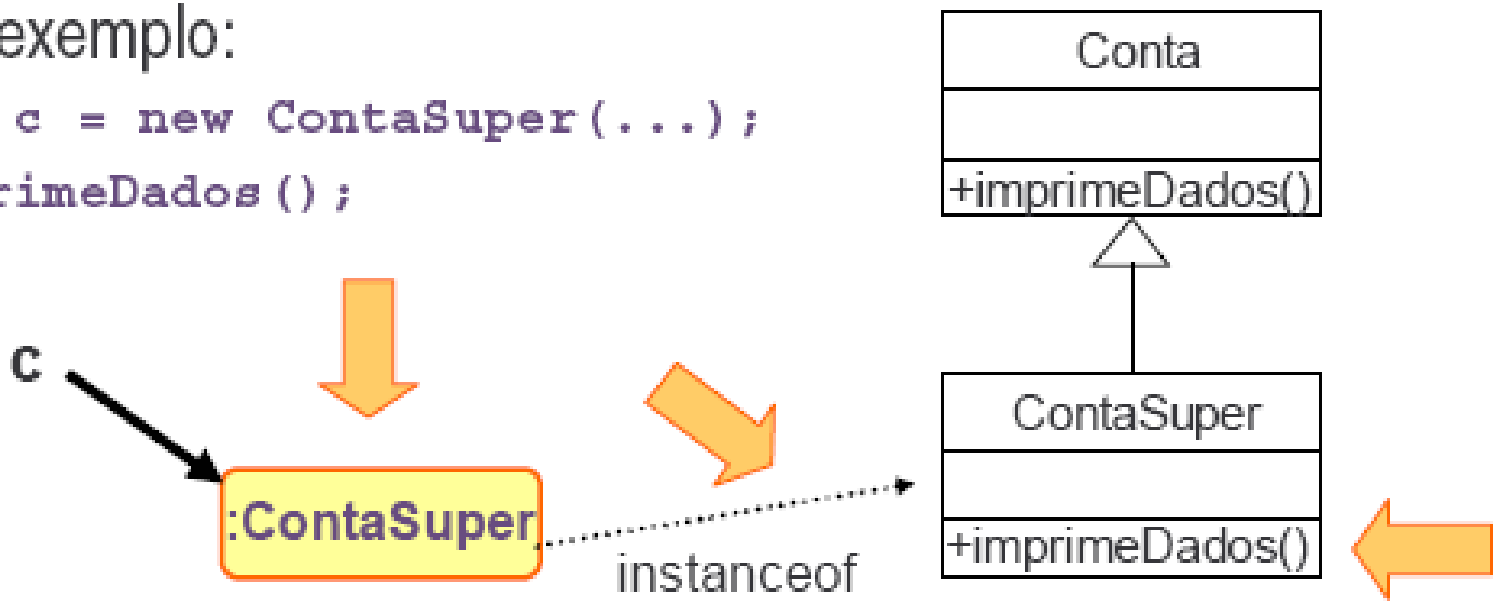


- imprimeDados
  - Em Conta e ContaCorrente, imprime o número da conta, o nome do titular e o saldo
  - Em ContaPoupança, imprime o número da conta, o nome do titular, o saldo e a data de aniversário da conta
  - Em ContaSuper, imprime o número da conta, o nome do titular, o saldo e o limite

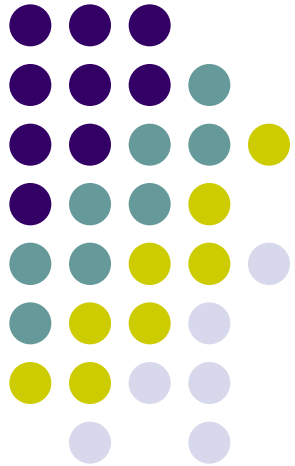
Exemplo de métodos polimórficos

- Em Java, podemos chamar o método `imprimeDados()` sobre uma referência para `Conta`, que será selecionado automaticamente o método correto das subclasses!
- Por exemplo:

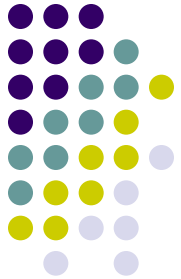
```
Conta c = new ContaSuper(...);
c.imprimeDados();
```



Classes e Métodos Abstratos

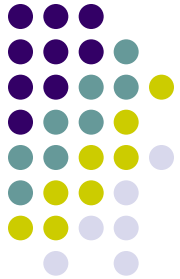


Classe Abstrata / Métodos abstratos



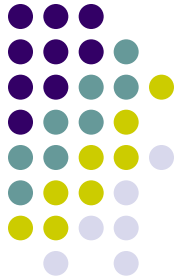
- Muitas classes abstratas são incompletas
  - Alguns ou todos os seus métodos não possuem implementação, servindo apenas para definir uma interface
    - Métodos abstratos

Classes abstratas na hierarquia de classes



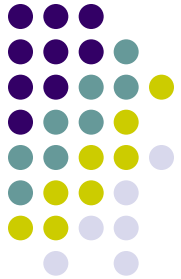
- Em uma hierarquia de classes, as classes que se encontram no topo da hierarquia são, geralmente, abstratas
  - Quanto mais alta a classe na hierarquia, mais abstrata é sua definição
- Uma classe no topo da hierarquia pode definir apenas o comportamento e os atributos que são comuns a todas as demais classes (suas subclasses)

Classe Abstrata (revisão)



- Classes abstratas **não** podem ser instanciadas
  - São utilizadas apenas para permitir a derivação de novas classes
  - Classes abstratas são identificadas pelo modificador abstract
    - public abstract class minhaClasse(){...}
  - Em uma classe abstrata, geralmente um ou mais métodos são declarados, mas não são implementados
    - Métodos abstratos

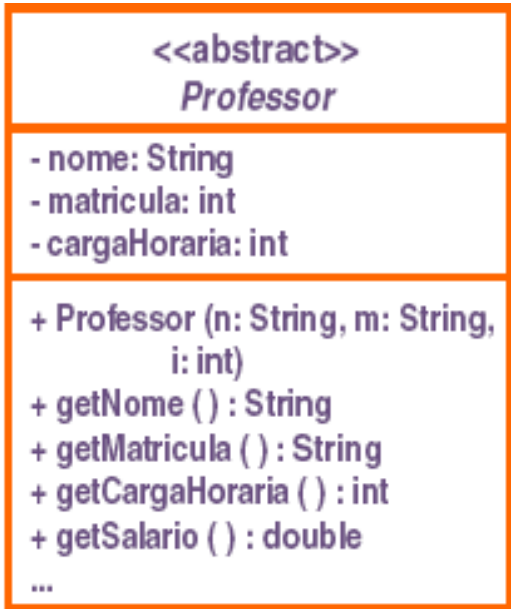
Métodos abstratos



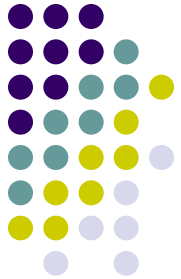
- São métodos sem código
  - São prefixados pela palavra abstract
  - Sua declaração termina com ";" após a declaração dos parâmetros
  - Um método abstrato indica que a classe não implementa aquele método e que ele deve ser obrigatoriamente implementado nas classes derivadas

Estudo de caso 2

- Considerando que todos os professores enquadram-se nas categorias de professor DE e professor horista, podemos definir a classe Professor como abstrata

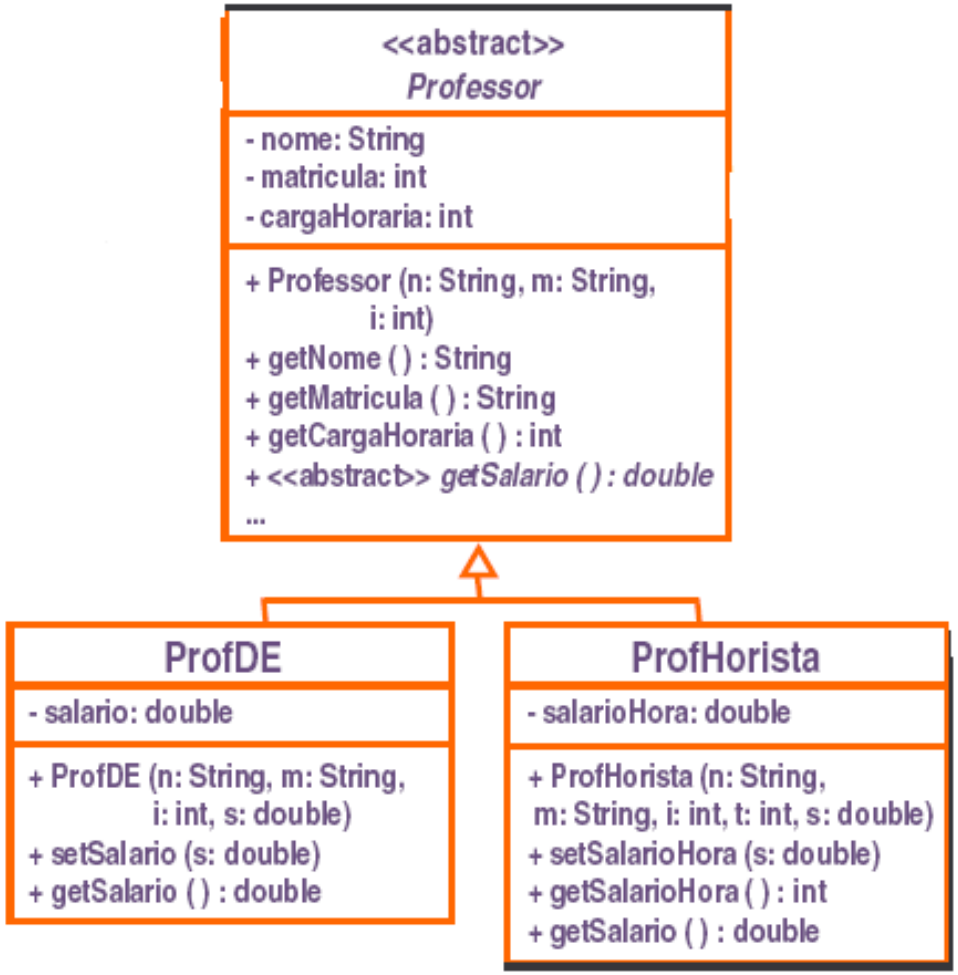


Estudo de caso 2

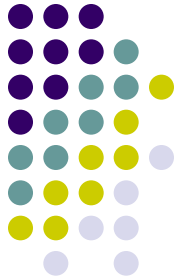


- Considere o método getSalario() da classe Professor
  - Sua implementação não faz sentido, pois o significado desse método estará definido nas subclasses
  - Logo, este método deve ser um método abstrato

Estudo de caso 2

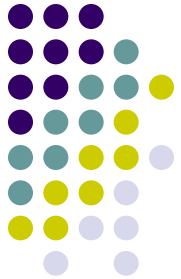


Exercício 2



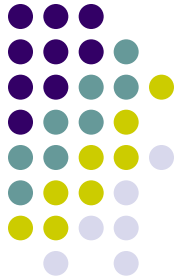
- Criar a classe abstrata Professor, com o método abstrato getSalario()
- Criar as classes concretas:
  - ProfessorDE -> salário mensal
  - ProfessorHorista -> valor da hora/aula, nro horas trabalhadas
- Criar um arrayList de Professor e inserir uma instância de cada tipo concreto
- Chamar, polimorficamente, o método getSalario()

Professor.java



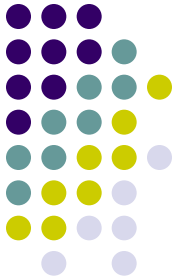
```
public abstract class Professor {
    private String nome;
    public Professor(String pNome) {
        nome = pNome;
    }
    public String getNome(){
        return nome;
    }
    public abstract double getSalario();
}
```

ProfessorDE.java



```
public class ProfessorDE extends Professor {
    private double salarioFixo;
    public ProfessorDE(String pNome, float pSalarioFixo) {
        super(pNome);
        salarioFixo = pSalarioFixo;
    }
    public double getSalario() {
        return salarioFixo;
    }
}
```

ProfessorHorista.java

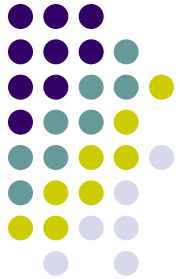


```
public class ProfessorHorista extends Professor {
    private double valorHoraAula;
    private int nroHorasTrab;
    public ProfessorHorista(String pNome, double pValorHoraAula,
                                          int pNroHorasTrab){
        super(pNome);
        valorHoraAula = pValorHoraAula;
        nroHorasTrab = pNroHorasTrab;
    }
    public double getSalario(){
        return (double)valorHoraAula * nroHorasTrab;
    }
}
```

TestaProfessor.java

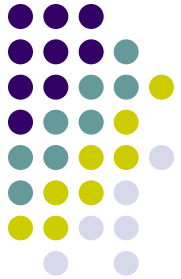
```
import java.util.*;

public class TestaProfessor {
    public static void main(String args[]){
        ArrayList<Professor> listaProf = new ArrayList<Professor>();
        listaProf.add(new ProfessorDE("Antonio Souza", 8000));
        listaProf.add(new ProfessorHorista("Marcelo Siqueira", 82.23, 64));
        for(Professor p: listaProf){
            System.out.println("O professor " + p.getNome() + " recebe R$ " +
                               p.getSalario() + " por mês");
        }
    }
}
```

Exercício 3
Entrega: 10/04



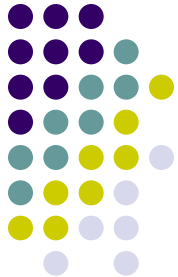
- Criar e manipular uma lista de professores utilizando um *ArrayList*
  - Tarefas:
    - Criar a lista
    - Utilizar JOptionPane para ler dados e construir instâncias de professoresDE e professores horistas
    - Exibir o nome de todos os professores
    - Calcular e exibir o salário dos professores com base na regra mostrada a seguir

Exercício 3
Entrega: 10/04



- Somente os professoresDE deverão recolher contribuição previdenciária, que corresponde a 11% do valor do salário.

Exercício 3
Entrega: 10/04



- Todos os professores devem ter o desconto do imposto de renda, conforme tabela a seguir
- Para ver como calcular:

https://www.mongeralaegon.com.br/blog/dinheiro/artigo/faixas-do-imposto-de-renda-qual-a-sua-aliquota-na-tabela-atualizada-do-ir-2016

Até 1.903,98	isento	isento
De 1.903,99 até 2.826,65	7,5%	142,80
De 2.826,66 até 3.751,05	15%	354,80
De 3.751,06 até 4.664,68	22,5%	636,13
Acima de 4.664,68	27,5%	869,36