Algoritmos e Programação II

Aula 5 – Herança

*Adaptado do material do Prof. Júlio Machado

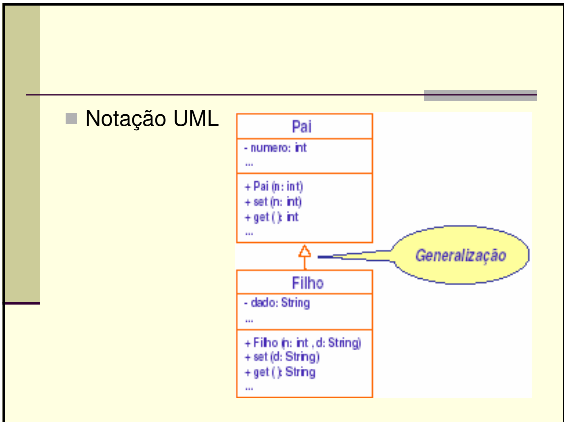
Herança

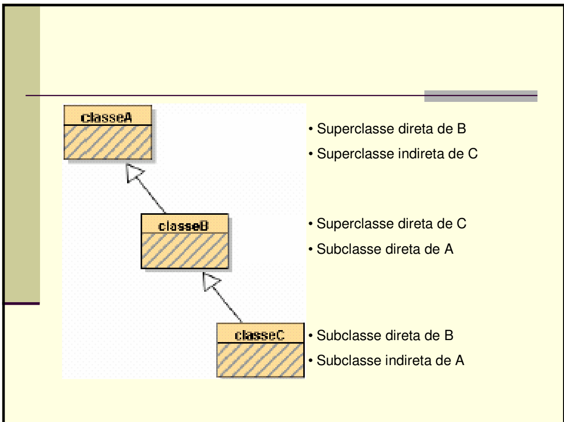
- Uma característica importante da programação orientada a objetos é permitir a criação de novas classes com base em uma classe já existente.
  - Esta classe pode ser tanto uma classe própria como uma classe padrão do Java, ou ainda uma classe construída por outra pessoa
    - Superclasse: classe já existente
- Subclasse: classe criada a partir da superclasse
- Objetivo: proporcionar o reuso de software.

- "Herança é a capacidade de reusar código pela **especialização** de soluções genéricas já existentes"
  - Novas classes são criadas a partir de outras já existentes
    - Subclasse herda de uma Superclasse
      - atributos
      - métodos
    - Subclasse
      - Absorve atributos e comportamentos além de adicionar os seus próprios
      - Pode sobrescrever métodos da superclasse

- Subclasses herdam de:
  - Superclasse direta - subclasse herda explicitamente
  - Superclasse indireta - subclasse herda de dois ou mais níveis superiores na hierarquia de classes
- Todo objeto da subclasse também é um objeto da superclasse, mas NÃO vice-versa
- A idéia na herança é "ampliar" a funcionalidade de uma classe
  - Subclasse contém tudo que a superclasse tem, além de novos atributos e métodos

- Notação UML





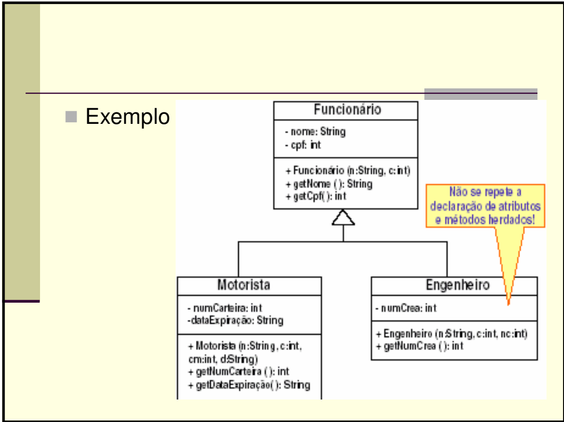
- Superclasse direta de B
- Superclasse indireta de C
- Superclasse direta de C
- Subclasse direta de A
- Subclasse direta de B
- Subclasse indireta de A

- Vocabulário:
  - Derivação: definição de uma nova classe baseada em outra já existente
  - Classe derivada: subclasse direta da classe da qual ela deriva
  - Classe base ou superclasse: classe original, que forma a base para a definição das classes derivadas
  - Superclasse de uma classe derivada: classe original

- Herança cria uma estrutura hierárquica
  - Exemplo
    - Hierarquia de classes para formas geométricas
    - Uma forma geométrica pode ser especializada em dois tipos:
      - bidimensional e tridimensional.

- Subclasse herda os membros da superclasse
  - São tratados como membros da subclasse, isto é, como se tivessem sido declarados dentro da subclasse.
  - Nem todos os atributos e métodos da superclasse são obrigatoriamente acessíveis na subclasse, pois isto dependerá dos modificadores de acesso.
  - Modificador **private** não permite acesso direto dentro da subclasse.
- Um objeto possui seus próprios atributos e métodos, mais os atributos e métodos da superclasse

- Exemplo



- Exemplo:

```
Funcionario func;
func = new Funcionario (“João”, 20338765210);
System.out.println(func.getNome());

Motorista mot;
mot = new
   Motorista(“José”,28765210635,00446677379,“18/02/2004”);
System.out.println(mot.getNome());
System.out.println(mot.getDataExpiracao());
```

func
nome : “João”
cpf : 20338765210

mot
nome : “José”
cpf : 28765210635
numCarteira : 00446677379
dataExpiracao: “18/02/2004”

- Utiliza-se a palavra-chave **extends** para definir herança de classes

```
class <subclasse> extends <superclasse> {
...
}
```

- Exemplo

```
public class Funcionario{...}
public class Motorista extends Funcionario{...}
public class Engenheiro extends Funcionario{...}
```

- Inclusão de membros da superclasse em uma classe derivada.

Observação: nem todos os membros da superclasse são obrigatoriamente acessíveis na classe derivada. Isto dependerá dos modificadores de acesso do membro.

- Atenção:
  - Atributos **private** da superclasse NÃO estão acessíveis diretamente na subclasse!
  - Para acessá-los diretamente na subclasse se pode utilizar o modificador de acesso **protected**

- Herdando atributos e métodos
  - **public** x **protected** x **private**
    - **public**: acessível em qualquer classe
    - **protected**: acessível por métodos da própria classe ou de uma subclasse
    - **private**: acessível somente nos métodos da própria classe
  - Métodos construtores NUNCA são herdados
    - Os métodos na classe derivada que foram herdados da superclasse, continuam podendo acessar TODOS os membros da superclasse

- Apesar dos construtores da superclasse não serem herdados, eles podem ser chamados para inicializar os atributos herdados (membros da superclasse), quando necessário
  - Utiliza-se **super()**
  - Se for chamado dessa forma, **deve ser o primeiro comando do construtor da subclasse**

- Exemplo anterior:

```
public class Funcionario {
  private String nome;
  private int cpf;
  public Funcionario(String n, int c) {
    nome = n;
    cpf = c;
  }
  public String getNome() {
     return nome;
  }
  public int getCpf() {
     return cpf;
   }
}
```

```
public class Motorista extends Funcionario {
   private int numCarteira;
   private String dataExpiracao;
   public int getNumCarteira() {
     return numCarteira;
   }
   public String getDataExpiracao() {
     return dataExpiracao;
   }
   public Motorista(String n, int c, int nc, String d)
    {
     nome = n;
     cpf = c;
     numCarteira = nc;
     dataExpiracao = d;
    }
}
```

Não podem ser acessados diretamento pois são private

- Subclasse tem acesso a todos os métodos da superclasse
  - Construtor da superclasse inicializa os atributos herdados:
    - **super()**
  - É obrigatória sua utilização!

```
public Motorista(String n, int c, int nc, String d) {
   super(n,c);
   numCarteira = nc;
   dataExpiracao = d;
}
```

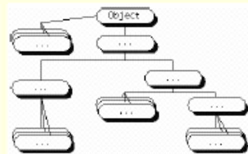
```
public class Engenheiro extends Funcionario {

   private int numCrea;

   public Engenheiro(String n, int c, int nc) {
      super(n,c);
      numCrea = nc;
   }

   public int getNumCrea() {
      return numCrea;
   }
}
```

- Hierarquia de Classes:
  - Em Java, todas as classes herdam diretamente ou indiretamente da classe **Object**
  - **Object** é o topo da hierarquia de classes em Java



- Hierarquia de Classes:
  - Toda classe criada sem explicitar uma superclasse, herda implicitamente da superclasse **Object**

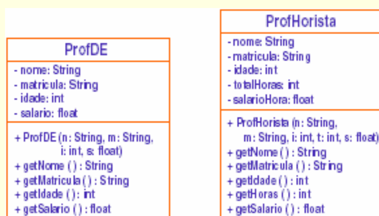
```
public class Ponto {
    ...
}
```

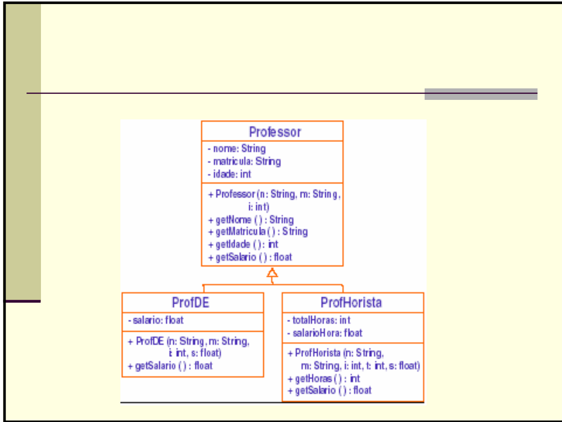
é equivalente a

```
public class Ponto extends Object {
    ...
}
```

- Estudo de caso
  - Os professores de uma universidade dividem-se em 2 categorias
    - professores em dedicação exclusiva (DE)
    - professores horistas
  - Professores em dedicação exclusiva possuem um salário fixo para 40 horas de atividade semanais
  - Professores horistas recebem um valor estipulado por hora
  - O cadastro de professores desta universidade armazena o nome, idade, matrícula e informação de salário

- Problema pode ser resolvido através de uma modelagem de classes como segue:






```
public class Professor
{
    protected String nome;
    protected int matricula;
    protected int cargaHoraria;
    public Professor(String n, int m, int c) {
      setNome(n);
      setMatricula(m);
      setCargaHoraria(c);
    }
    public void setNome(String n) { nome = n; }
    public String getNome() { return nome; }
    public void setMatricula(int m) { matricula = m; }
    public int getMatricula() { return matricula; }
    public void setCargaHoraria(int c) { cargaHoraria = c; }
    public int getCargaHoraria() { return cargaHoraria; }
    public double getSalario() { return 0; }
}
```

```
public class ProfDE extends Professor
{
    private double salario;
    public ProfDE(String n, int m, double s) {
            super(n,m,40);
            setSalario(s);
    }
    public void setSalario(double s) {
            salario = s;
    }
    public double getSalario() {
            return salario;
    }
}
```

```
public class ProfHorista extends Professor
{
    private double salarioHora;
    public ProfHorista(String n, int m, int c, double s) {
      super(n,m,c);
      setSalarioHora(s);
    }
    public void setSalarioHora(double s) {
      salarioHora = s;
    }
    public double getSalarioHora() {
      return salarioHora;
    }
    public double getSalario() {
      return (salarioHora * cargaHoraria * 4.5);
    }
}
```

Modificadores de Acesso

- Relembrando: classes são agrupadas em conjuntos chamados ***packages*** (=pacotes), sendo que cada um é guardado em um diretório diferente.
  - Exemplo: o pacote **java.lang** está armazenado no diretório **java/lang**.
  - Para usar um pacote:
    import java.util.*;

- Modificador Acesso permitido…
  - sem modificador - de todas as classes no mesmo pacote
  - **public -** de qualquer classe em qualquer pacote
  - **private -** sem acesso de fora da classe
  - **protected -** de todas as classes no mesmo pacote e a partir de qualquer subclasse em qualquer pacote
  - ATENÇÃO:
    - Para derivar uma classe de fora do pacote contendo a superclasse, a superclasse deve ser declarada como **public**.

- Recomendações
  - Métodos que fazem a interface externa de uma classe devem ser declarados como **public** (sendo portanto herdados pelas subclasses)
  - Atributos devem ser **private**.
  - Utiliza-se o atributo **protected** quando classes são definidas dentro de um pacote e se deseja dar ao usuário do pacote (desenvolvedor de classes em outro pacote) acesso apenas às subclasses
  - Classes, em geral, são declaradas como **public**
- Exceções:
  - Atributos de uma classe definidos como **final** possuem valor fixo e podem ser usados de fora da classe. Assim sendo, recomenda-se que sejam declarados como **public**.
  - Métodos que devem ser usados somente dentro da própria classe, devem ser especificados como **private**.

- Notação UML:
  - *private*: **-**
  - *protected*: **#**
  - *public*: **+**
- Observações:
  - atributo **de classe** é sublinhado
  - método **de classe** é sublinhado

Sobreescrita de Métodos

- Uma subclasse pode sobrescrever (*override*) métodos da superclasse.
  - Sobrescrita permite completar ou modificar um comportamento herdado.
  - Quando um método é referenciado em uma subclasse, a versão escrita para a subclasse é utilizada, ao invés do método na superclasse.
  - É possível acessar o método original da superclasse:

    super.nomeDoMetodo()

- Cuidado:
  - O tipo de retorno, nome do método, número e tipo dos parâmetros do método da subclasse deve ser o mesmo do método da superclasse.
  - O modificador de acesso do método da subclasse pode relaxar o acesso, mas não o contrário.
    - Ex.: um método **protected** na superclasse pode ser tornado **public** na subclasse mas não **private**.
  - Uma subclasse não pode sobrescrever um método de classe da superclasse.

Exercícios

1)Codifique em Java a seguinte hierarquia de classes para objetos geométricos:
  - Classe Ponto
    - Atributos: x, y
    - Métodos: setX, setY, getX, getY
  - Classe Circulo (extends Ponto)
    - Atributo: raio
    - Métodos: setRaio, getRaio, getArea
  - Classe Cilindro (extends Circulo)
    - Atributo: altura
    - Métodos: setAltura, getAltura, getArea (superfície), getVolume
  - Obs: para todas as classes devem ser implementados dois construtores, um com parâmetros para inicializar os atributos e outro sem parâmetros

2)Defina uma classe em Java chamada "Produto" para armazenar as informações de um produto. A classe deve armazenar o nome do produto, a quantidade armazenada e o preço unitário. A rotina construtora deve receber o nome, a quantidade e o preço unitário. A classe deve oferecer rotinas tipo "get" para todos os campos. Deve oferecer ainda uma rotina onde se informa uma certa quantidade a ser retirada do estoque e outra onde se informa uma certa quantidade a ser acrescida ao estoque. A rotina onde se informa uma quantidade a ser retirada do estoque deve retornar a quantidade que efetivamente foi retirada (para os casos em que havia menos produtos do que o solicitado).

3)Defina uma classe em Java derivada da classe "Produto" chamada "ProdutoPerecivel" que possui um atributo extra que guarda a data de validade do produto. As rotinas através das quais se informa as quantidades a serem retiradas ou acrescidas do estoque devem ser alteradas. A rotina de retirada deve receber também por parâmetro a data do dia corrente. Se os produtos já estiverem armazenados há mais de 2 meses a rotina deve zerar o estoque e devolver 0, pois produtos vencidos são jogados fora. A rotina de acréscimo no estoque só deve acrescentar os novos produtos caso o estoque esteja zerado, de maneira a evitar misturar produtos com prazos de validade diferenciados.

4) Defina uma classe Java derivada da classe "ProdutoPerecivel" chamada "ProdPerEsp" que oferece uma rotina de impressão de dados capaz de imprimir uma nota de controle onde consta o nome do produto, a quantidade em estoque e a data de validade.

5) Defina uma classe "ProdutoComPreco" derivada de "Produto" que possua campos para armazenar o preço unitário do produto. A classe deve oferecer rotinas para permitir obter o preço unitário, alterar o preço unitário (sempre positivo).