Métodos e Classes Abstratas

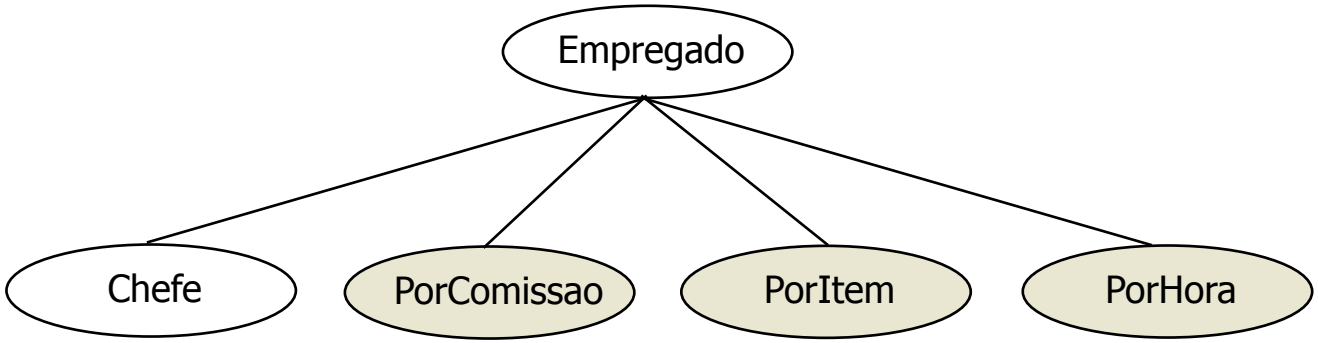
- Como vimos, subclasses podem redefinir (@Override) um método definido em sua superclasse. Assim como podem usar o método da forma como foi herdado da superclasse.
- Para indicar que um método de uma classe deve ser necessariamente redefinido em cada uma de suas subclasses este método deve ser declarado como abstract.
- O que é um método abstrato? É um método que não tem um corpo, ou seja, um método não implementado.

```
public void mostrar()
{
  System.out.println(nome + ' ' + sobrenome);
}

public abstract double ganha();
```

- Uma classe que contém um ou mais métodos abstratos deve ser declarada explicitamente como abstrata. Essa classe, no entanto, pode ter métodos concretos (não-abstratos).

5. Classes Abstratas e Interfaces

- Não é possível criar objetos de uma classe abstrata.
- Para que serve uma classe para a qual não se pode criar objetos?
- Se uma subclasse é derivada de uma superclasse que contém um método abstrato e se esse método abstrato não for redefinido na subclasse, esse método permanece abstrato na subclasse. Com isso, a subclasse também deverá ser declarada explicitamente como abstract.



- Declarar um método como abstrato é uma forma de obrigar o programador a redefinir esse método em todas as subclasses para as quais deseja criar objetos.

5. Classes Abstratas e Interfaces

Exemplo:

Empregado.java

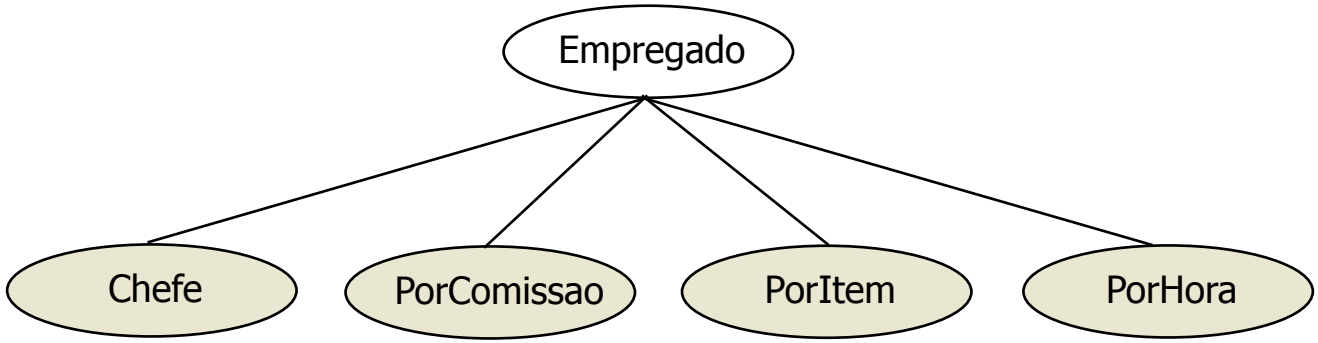
```
// Classe abstrata Empregado

public abstract class Empregado
{
  private String nome;
  private String familia;

  public Empregado(String n, String f)
  {
    nome = n;
    familia = f;
  }

  public String getNome() { return nome; }

  public String getFamilia() { return familia; }

  public String toString()
  {
    return nome + ' ' + familia;
  }

  public abstract double ganha();
}
```

Como a classe contém um método abstrato, ela deve ser declarada como abstrata.

Método abstrato. Deve ser implementado nas subclasses.

5. Classes Abstratas e Interfaces

- Seja a hierarquia de classes:



onde Chefe, PorComissao, PorItem e PorHora são classes finais.

- Todas essas subclasses vão precisar redefinir o método ganha(). Como se tratam de tipos diferentes de empregado, cada um ganha de uma forma:
  - Chefe: salário fixo e predefinido;
  - PorComissao: valor fixo + comissão * vendas;
  - PorItem: valor por produção * quantidade produzida;
  - PorHora: valor por hora * total de horas trabalhadas.
- Declarando o método ganha() como abstrato na superclasse garante-se que nas 4 subclasses haverá a implementação do método ganha() para cada tipo de empregado (do contrário, para que servem as subclasses?).

Chefe.java

```
public final class Chefe extends Empregado
{
  private double salario;

  public Chefe(String n, String f, double s)
  {
    super(n,f);
    setSalario(s);
  }

  public void setSalario( double s )
  {
    salario = (s > 0 ? s : 0.0);
  }

  public double ganha()
  {
    return salario;
  }

  public String toString()
  {
    return "Chefe: " + super.toString();
  }
}
```

Implementação do método abstrato da superclasse.

5. Classes Abstratas e Interfaces

PorComissao.java

```
public final class PorComissao extends Empregado
{
  private double salario;
  private double comissao;  // comissao por item vendido
  private int vendas;       // numero de itens vendidos

  public PorComissao(String n, String f, double s, double c, int v)
  {
    super(n,f);
    setSalario(s);
    setComissao(c);
    setVendas(v);
  }

  public void setSalario(double s) { salario = (s > 0 ? s : 0.0); }
  public void setComissao(double c) { comissao = (c > 0 ? c : 0.0); }
  public void setVendas(int v) { vendas = (v > 0 ? v : 0); }

  public double ganha() {
    return salario + comissao*vendas;
  }

  public String toString() {
    return "Por comissao: " + super.toString();
  }
}
```

Implementação do método abstrato da superclasse.

5. Classes Abstratas e Interfaces

PorItem.java

```
public final class PorItem extends Empregado
{
  private double producao;  // salario por producao
  private int quantidade;   // quantidade produzida

  public PorItem(String n, String f, double p, int q)
  {
    super(n,f);
    setProducao(p);
    setQuantidade(q);
  }

  public void setProducao(double p) { producao = (p > 0 ? p : 0.0); }
  public void setQuantidade(int q) { quantidade = (q > 0 ? q : 0); }

  public double ganha()
  {
    return producao*quantidade;
  }

  public String toString()
  {
    return "Por item: " + super.toString();
  }
}
```

Implementação do método abstrato da superclasse.

PorHora.java

```
public final class PorHora extends Empregado
{
  private double valor;     // salario por hora
  private double horas;     // horas trabalhadas (300 horas no máximo)

  public PorHora(String n, String f, double v, double h)
  {
    super(n,f);
    setValor(v);
    setHoras(h);
  }

  public void setValor(double v) { valor = (v > 0 ? v : 0.0); }
  public void setHoras(double h)
  {
    horas = (h >= 0 && h <= 300 ? h : 0.0);
  }

  public double ganha()
  {
    return valor*horas;
  }

  public String toString() {
    return "Por hora: " + super.toString();
  }
}
```

Implementação do método abstrato da superclasse.

5. Classes Abstratas e Interfaces

Teste1.java

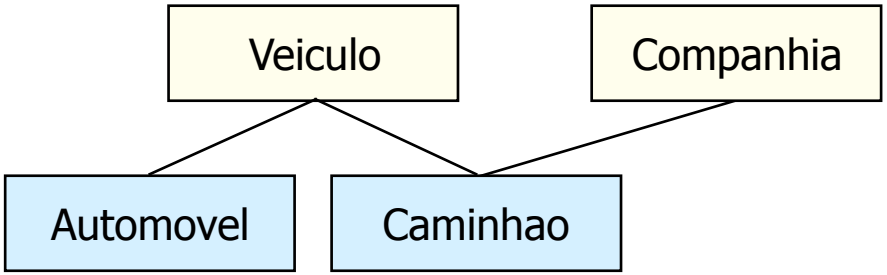
```
import java.text.DecimalFormat;

public class Teste1
{
  public static void main( String args[] )
  {
    DecimalFormat df = new DecimalFormat("0.00");

    Chefe ch = new Chefe("Joao","Silva",3000.00);
    PorComissao pc = new PorComissao("Maria","Souza",400.00,3.00,150);
    PorItem pi = new PorItem("Pedro","Cabral",2.50,200);
    PorHora ph = new PorHora("Marta","Ferreira",13.75,40.50);

    System.out.println(ch.toString() + " ganha $" + df.format(ch.ganha()));
    System.out.println(pc.toString() + " ganha $" + df.format(pc.ganha()));
    System.out.println(pi.toString() + " ganha $" + df.format(pi.ganha()));
    System.out.println(ph.toString() + " ganha $" + df.format(ph.ganha()));
  }
}
```

```
Chefe: Joao Silva ganha $3000,00
Por comissao: Maria Souza ganha $850,00
Por item: Pedro Cabral ganha $500,00
Por hora: Marta Ferreira ganha $556,88
```

Interfaces

- Classes abstratas contém, pelo menos, um método abstrato mas podem conter campos e também métodos concretos.
- Interfaces são classes abstratas que contêm apenas métodos abstratos. Portanto, em uma interface não existe implementação alguma. A definição de uma interface é feita pela palavra-chave interface (em vez de class).
- Para utilizar uma interface, uma classe deve especificar que implementa (implements) a interface e a classe deve definir cada método da interface respeitando o número de argumentos e o tipo de retorno especificado na interface. Se a classe deixar de implementar algum método da interface, a classe se torna abstrata.
- Uma classe pode implementar mais de uma interface.
- Uma mesma interface pode ser implementada por várias classes.

Exemplo:



- **Veiculo** e **Companhia** são interfaces; **Automovel** e **Caminhao** são classes.
- Note que a classe **Caminhao** implementa as duas interfaces. Se Veiculo e Companhia fossem classes, Caminhao não poderia ser subclasse de ambas, pois haveria duas superclasses.
- Note que a interface **Veiculo** é implementada por duas classes: Automovel e Caminhao.
- A principal razão para a utilização de interfaces é declarar métodos que uma ou mais classes devem necessariamente implementar (pois, do contrário, se tornariam classes abstratas).

5. Classes Abstratas e Interfaces

- Uma interface pode ser usada como uma forma de apresentar aos usuários os métodos disponíveis, sem revelar os detalhes de implementação desses métodos.
- Isso corresponde à interface pública de uma classe.
- Outra utilização de interfaces é na definição de constantes, ou seja, campos public static final. Exemplo:

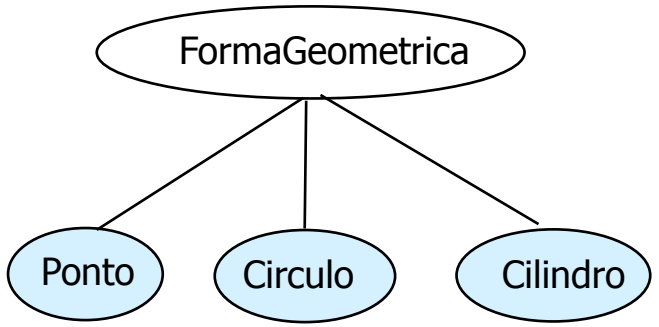
Constantes.java

```
public interface Constantes
{
  public static final int UM = 1;
  public static final int DOIS = 2;
  public static final int TRES = 3;
}
```

- Note que, como não há métodos declarados nessa interface, uma classe que implementa essa interface não é obrigada a fornecer implementação alguma.
- Uma classe que implementa a interface Constantes pode utilizar as constantes UM, DOIS e TRES em qualquer lugar da classe.

5. Classes Abstratas e Interfaces

- Exemplo: Desejamos que as classes Ponto, Circulo e Cilindro disponham dos métodos forma() e mostrar().




```
public interface FormaGeometrica
{
  public String forma();
  public void mostrar();
}
```

```
public class Ponto implements FormaGeometrica
{
  private double coordx;
  private double coordy;

  public Ponto(double x, double y) {
    coordx = x;
    coordy = y;
  }

  public String forma() {
    return "Ponto: (" + coordx + ", " + coordy + ")";
  }

  public void mostrar() {
    System.out.println(forma());
  }
}
```

5. Classes Abstratas e Interfaces

```
public class Circulo implements FormaGeometrica
{
  private Ponto centro;
  private double raio;

  public Circulo(Ponto p, double r)
  {
    centro = p;
    raio = r;
  }

  public String forma()
  {
    return "Circulo: centro = " + centro.forma() + "; raio = " + raio;
  }

  public void mostrar()
  {
    System.out.println(forma());
  }
}
```

5. Classes Abstratas e Interfaces

```
public class Cilindro implements FormaGeometrica
{
  private Circulo base;
  private double altura;

  public Cilindro(Circulo b, double a)
  {
    base = b;
    altura = a;
  }

  public String forma()
  {
    return "Cilindro: base = " + base.forma() + "; altura = " + altura;
  }

  public void mostrar()
  {
    System.out.println(forma());
  }
}
```

- Notou que o método mostrar() é igual em todas as classes?
- Como evitar que esse método seja implementado em todas as classes?

5. Classes Abstratas e Interfaces

- Basta considerar FormaGeometrica como classe abstrata e não como interface. O método mostrar() pode ser implementado na superclasse e herdado pelas subclasses.

```
public abstract class FormaGeometrica
{
  public abstract String forma();
  public void mostrar()
  {
    System.out.println(forma());
  }
}
```

```
public class Ponto extends FormaGeometrica {
  private double coordx;
  private double coordy;

  public Ponto(double x, double y) {
    coordx = x;
    coordy = y;
  }

  public String forma() {
    return "Ponto: (" + coordx + ", " + coordy + ")";
  }
}
```

5. Classes Abstratas e Interfaces

Teste2.java

```
public class Teste2
{
  public static void main(String[] args)
  {
    Ponto p = new Ponto(2,3);
    Circulo c = new Circulo(p,10);
    Cilindro d = new Cilindro(c,5);
    p.mostrar();
    c.mostrar();
    d.mostrar();
  }
}
```

```
Ponto: (2.0, 3.0)
Circulo: centro = Ponto: (2.0, 3.0); raio = 10.0
Cilindro: base = Circulo: centro = Ponto: (2.0, 3.0); raio = 10.0; altura = 5.0
```