PROGRAMAÇÃO ORIENTADA A OBJETOS

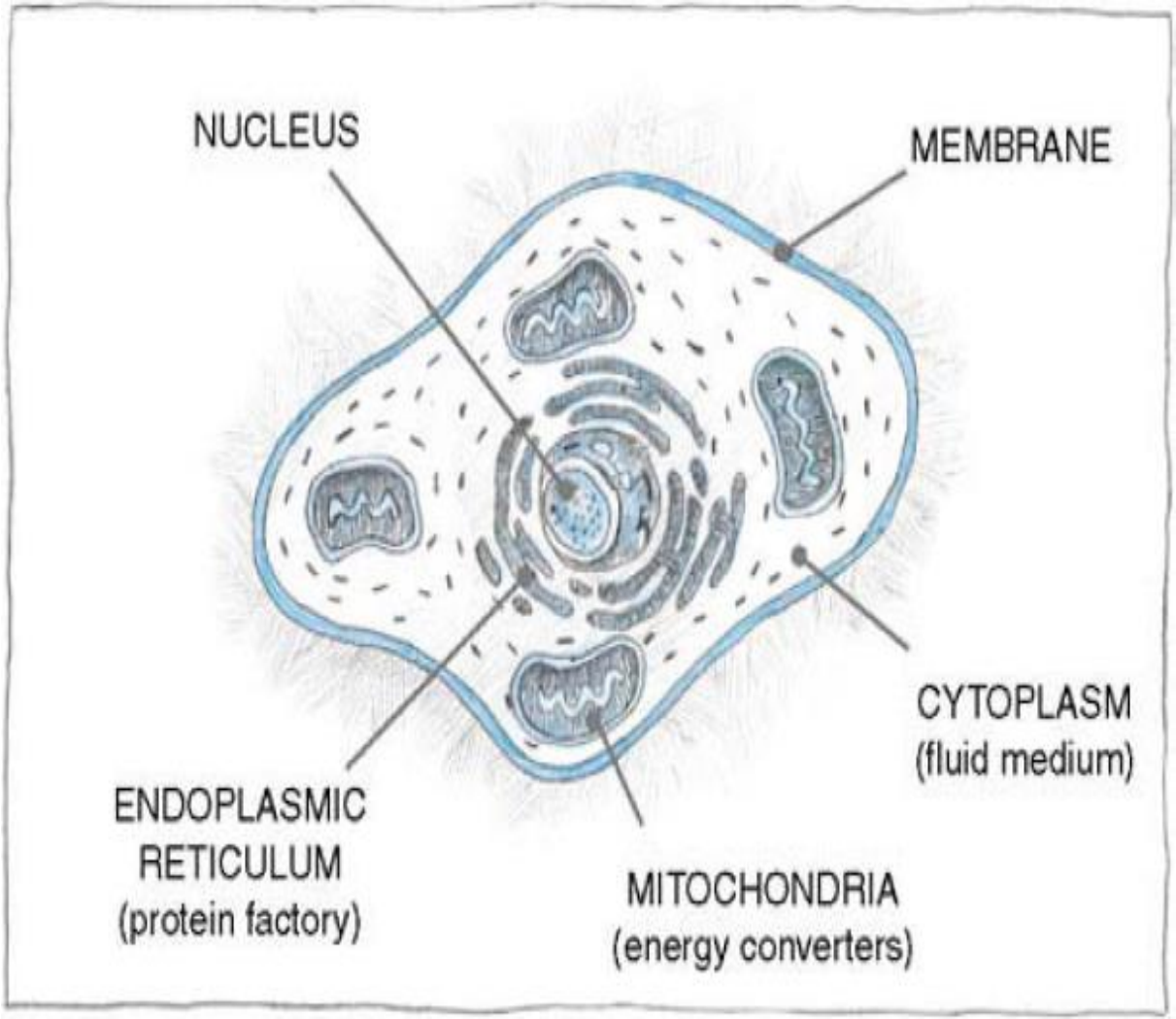
Introdução a Herança e Encapsulamento

ENCAPSULAMENTO

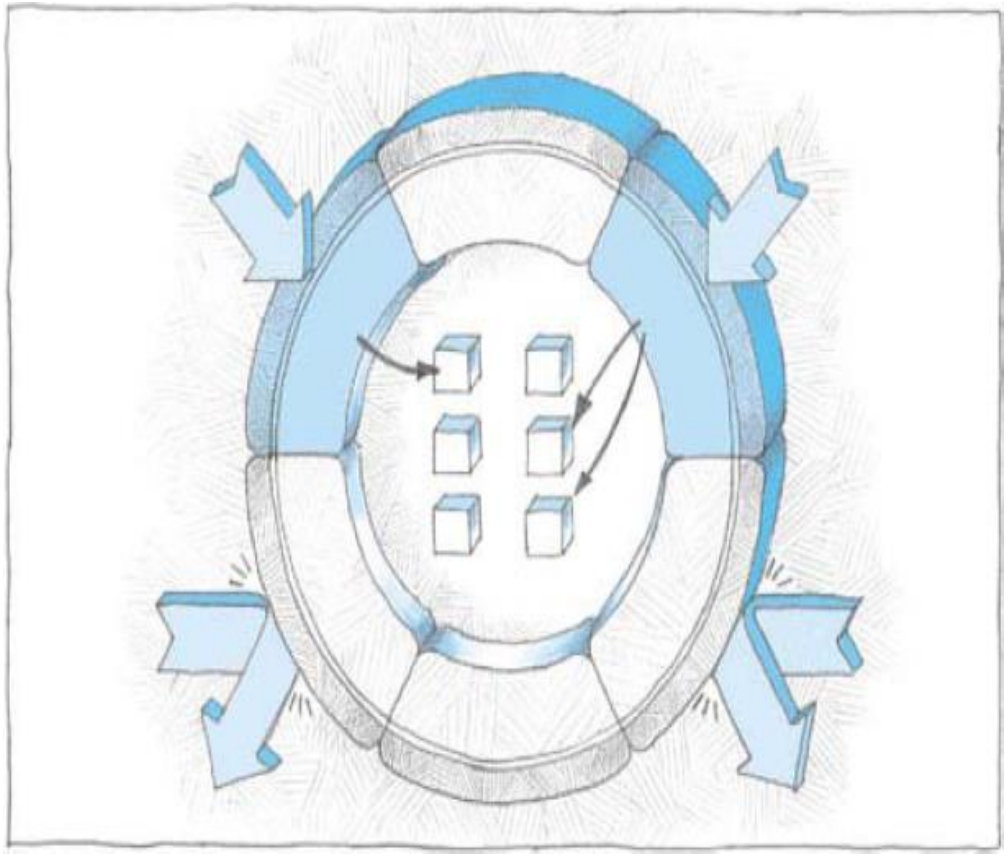
Comparando com a Biologia

- Todas as interações entre células ocorrem através de mensagens químicas reconhecidas pela membrana da célula e passadas para o interior da célula.
- Esta membrana protege o funcionamento interno da célula da intromissão externa e também oculta a complexidade da célula oferecendo uma interface relativamente simples para o restante do organismo.
- Esse mecanismo de comunicação baseado em mensagens simplifica enormemente o funcionamento das células. As células não tem de ler as moléculas de proteínas umas das outras para obter o que elas precisam. Tudo o que elas têm de fazer é enviar a mensagem química apropriada e a célula receptora responde de forma apropriada.

Comparando com a Biologia



Encapsulamento – a interface



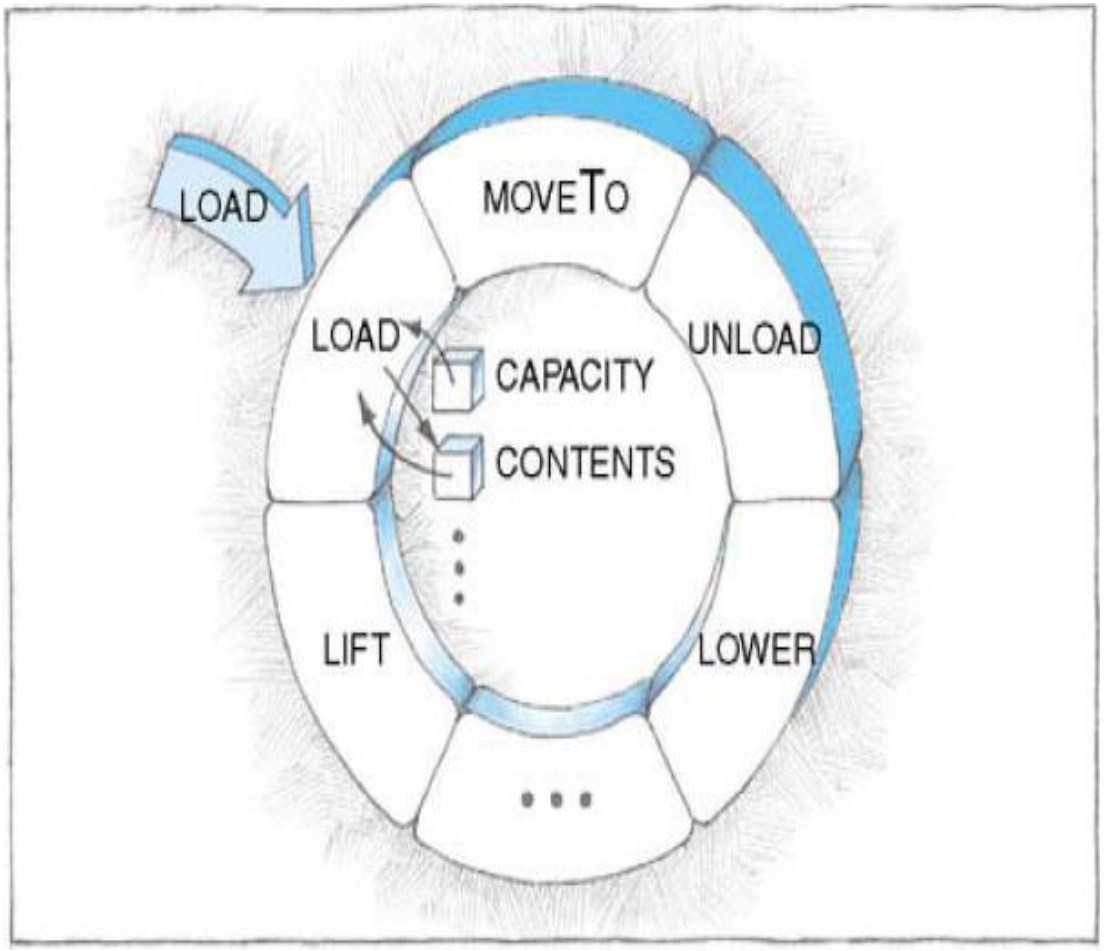
Interface segundo o dicionário:
Aquilo que ocasiona uma união física ou lógica entre dois sistemas que, diretamente, não poderiam estar conectados.

Interface em tecnologia:
Circuito, dispositivo ou porta que permite que duas ou mais unidades incompatíveis sejam interligadas num sistema padrão de comunicação, permitindo que se transfiram dados entre eles.

Encapsulamento – a interface

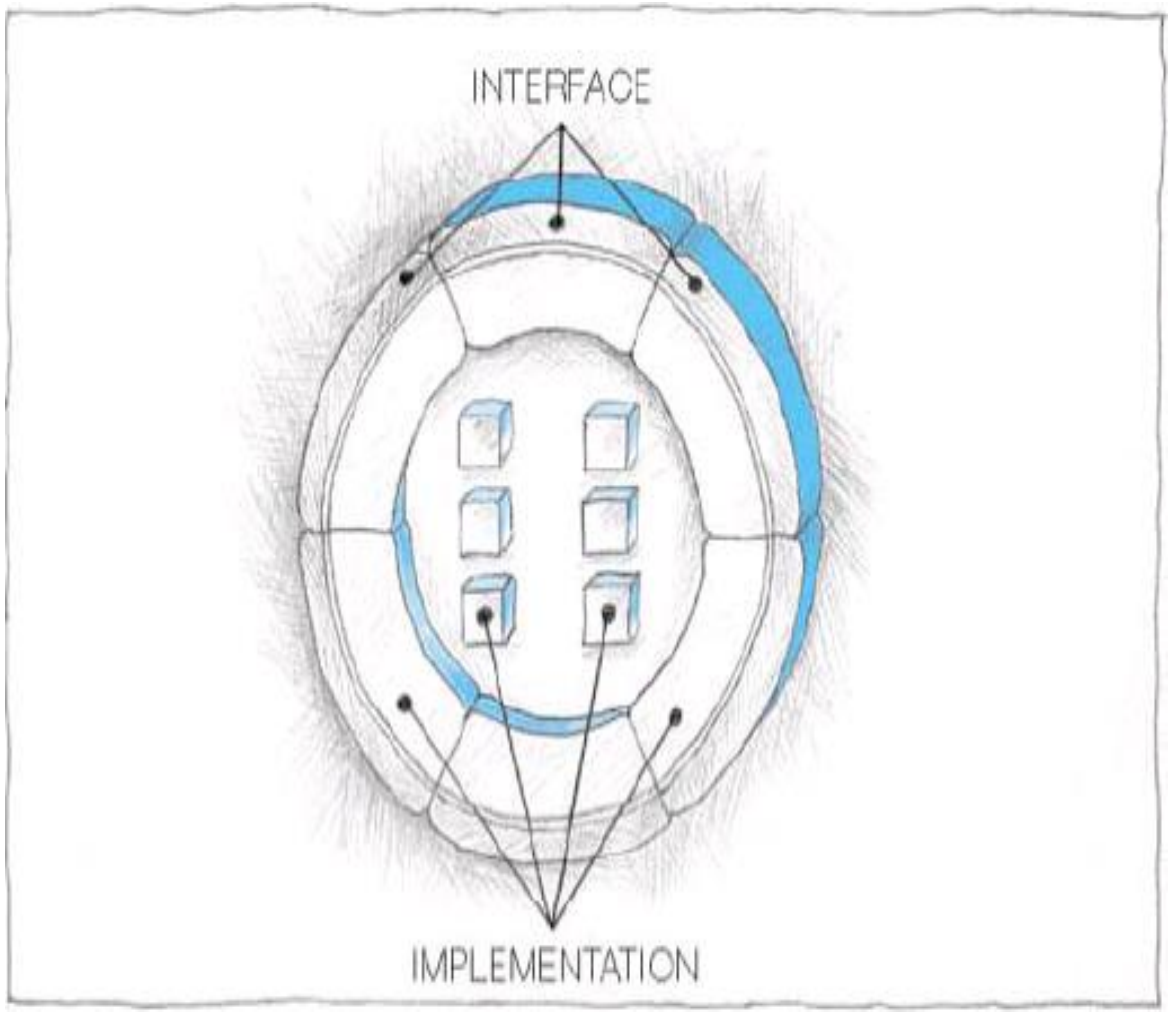
- A chave para o encapsulamento dos objetos é a interface de mensagens.
- A interface do objeto cumpre o mesmo papel que a membrana da célula: levantar uma barreira entre a estrutura interna do objeto e tudo o que reside fora do objeto.
- De forma análoga à membrana da célula, a interface dos objetos assegura que todas as interações com o objeto se darão através de um sistema pré-definido de mensagens que o objeto entende e trata corretamente.

Encapsulamento – a interface



Somente os métodos da classe podem acessar os atributos de um objeto

Encapsulamento – implementação da interface



Encapsulamento

- “Encapsulamento é o processo de esconder detalhes de implementação de uma classe, deixando para o seu usuário apenas uma interface de interação.”

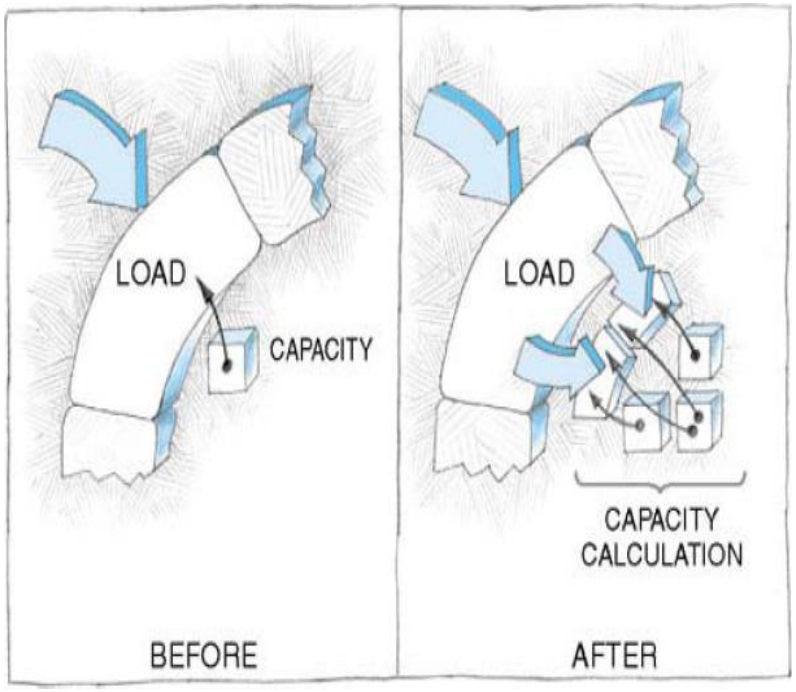
- A interface protege os componentes internos de um objeto de serem corrompidos por outros objetos.

- No sentido oposto, a interface de um objeto protege os outros objetos contra variações nos componentes internos deste mesmo objeto.

Encapsulamento

- Assim a interface de um objeto é o conjunto de métodos públicos que ele possui.
- Suponha que a capacidade do veículo, antes armazenada em uma variável, agora tenha de ser calculada através da interação de uma série de métodos
- O encapsulamento é obtido através dos modificadores de visibilidade

Encapsulamento – aplicação independente da implementação da interface



Proteção Contra Variações

- Apesar da implementação ter mudado, como o cliente é cliente da interface, não da implementação, o projeto está protegido contra variações.
- Assim, dizemos que, sempre que possível, devemos programar para interface não para a implementação

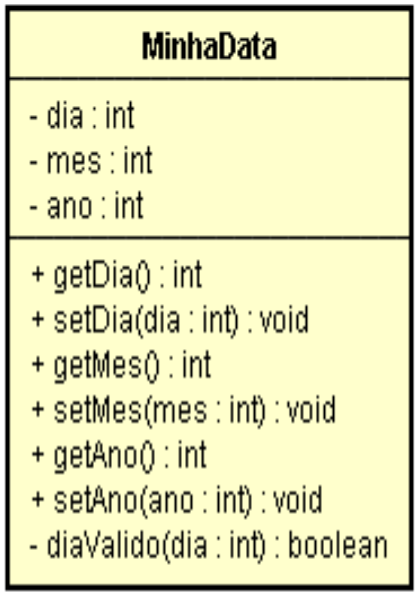
Encapsulamento - implementação

- O problema, o código do cliente tem acesso direto aos dados do objeto


```
class Teste {
    public static void main(String[] args) {
        MinhaData d = new MinhaData();

        d.dia = 32; // erro!

        d.mes = 2;
        d.dia = 30; // erro!

        d.dia = d.dia + 1;
        // pode dar erro!
    }
}
```

Encapsulamento - implementação

- A solução: O código do cliente deve usar os setters e getters para acessar os dados internos do objeto




```
MinhaData d = new MinhaData();

d.setDia(32);
// dia inválido, retorna false

d.setMes(2);
d.setDia(30);
// erro! setDia retorna false

d.setDia(d.getDia()+1);
// retorna false se último dia do mês
```

Verifica o número de dias no mês

Encapsulamento - Visibilidade

- Inicialmente, a visibilidade dos métodos de acesso deve ser mantida em **protected** de modo que somente as subclasses possam usá-los, ou **private** restringindo o acesso a apenas a própria classe.

- Somente torne o método de acesso público quando um objeto externo precisar acessar o atributo.

Encapsulamento – vantagens dos métodos de acesso

- **Atualização de atributos**: um único ponto para atualização de atributos tornando mais fáceis as modificações e testes

- **Redução do acoplamento entre a superclasse e suas subclasses**

- **Encapsular modificações nos atributos:** Você pode modificar um método de acesso de modo a fornecer aos clientes da classe a mesma funcionalidade anterior à modificação

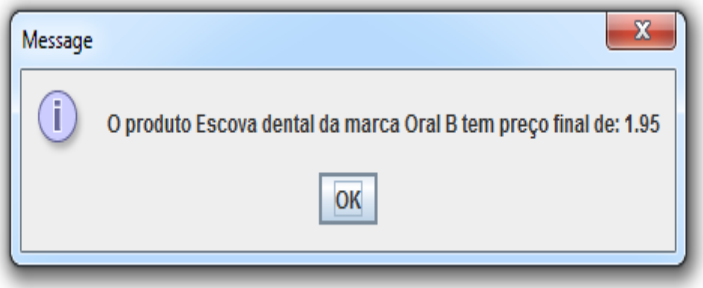
Encapsulamento – Outros exemplos

```
public class Produto {

    public String nome, codigo, marca;
    public double valorCusto;
    public double valorVenda;

    public void calculaValorVenda(double taxa){

        valorVenda = valorCusto + (valorCusto * (taxa/100));
    }

}
```

Acesso public nos atributos da classe, ou seja, a classe cliente (que criará e utilizará um objeto Produto) pode acessar diretamente os atributos da classe Produto

Encapsulamento – Outros exemplos

```
import javax.swing.JOptionPane;
public class ProgramaEstoque {

    public static void main (String [] args){

        Produto novoProduto = new Produto();

        novoProduto.nome = "Escova dental";
        novoProduto.codigo = "011";
        novoProduto.marca = "Oral B";
        novoProduto.valorCusto = 1.50;

        double taxaCalculo = Double.parseDouble(JOptionPane.showInputDialog("Digite a taxa a ser aplicada no valor final:"));
        novoProduto.calculaValorVenda(taxaCalculo);

        JOptionPane.showMessageDialog(null, "O produto " +novoProduto.nome + " da marca "
                                      +novoProduto.marca +" tem preço final de: " +novoProduto.valorVenda);

    }

}
```

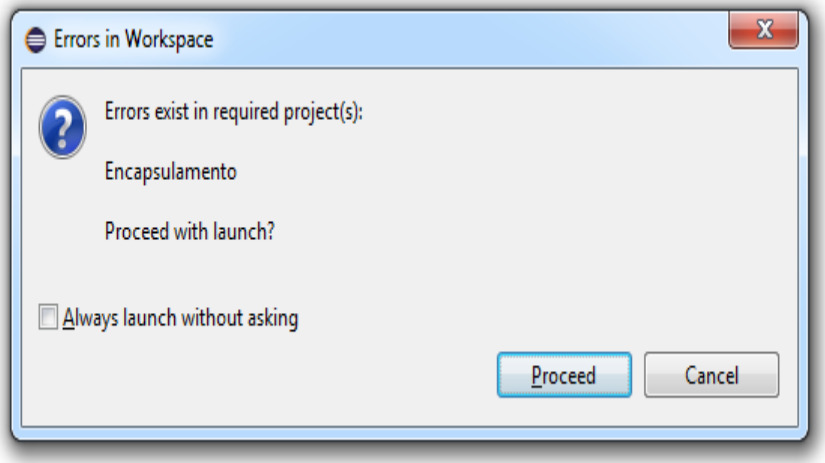


Esse programa rodará sem erros, pois os atributos da classe Produto são públicos

Encapsulamento – Outros exemplos

```
public class Produto {

    private String nome, codigo, marca;
    private double valorCusto;
    private double valorVenda;

    public void calculaValorVenda(double taxa){

        valorVenda = valorCusto + (valorCusto * (taxa/100));
    }

}
```

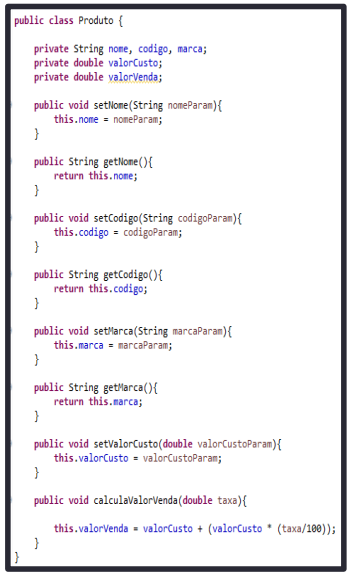
Mas se tornarmos agora os atributos como privados, veja o que acontece ao executar o programa principal




```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field Produto.nome is not visible
        The field Produto.codigo is not visible
        The field Produto.marca is not visible
        The field Produto.valorCusto is not visible
        The field Produto.nome is not visible
        The field Produto.marca is not visible
        The field Produto.valorVenda is not visible

        at ProgramaEstoque.main(ProgramaEstoque.java:8)
```

Encapsulamento – Outros exemplos

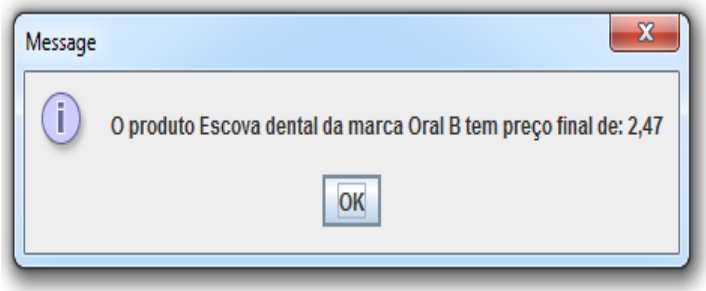
Devemos criar métodos (estes públicos) que permitam manipular os atributos.



```
public class Produto {

    private String nome, codigo, marca;
    private double valorCusto;
    private double valorVenda;

    public void setNome(String nomeParam){
        this.nome = nomeParam;
    }

    public String getNome(){
        return this.nome;
    }

    public void setCodigo(String codigoParam){
        this.codigo = codigoParam;
    }

    public String getCodigo(){
        return this.codigo;
    }

    public void setMarca(String marcaParam){
        this.marca = marcaParam;
    }

    public String getMarca(){
        return this.marca;
    }

    public void setValorCusto(double valorCustoParam){
        this.valorCusto = valorCustoParam;
    }

    public void calculaValorVenda(double taxa){

        this.valorVenda = valorCusto + (valorCusto * (taxa/100));
    }
}
```

Encapsulamento – Outros exemplos

```
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
public class ProgramaEstoque {

    public static void main (String [] args){

        Produto novoProduto = new Produto();

        novoProduto.setNome("Escova dental");
        novoProduto.setCodigo("011");
        novoProduto.setMarca("Oral B");
        novoProduto.setValorCusto(1.90);

        double taxaCalculo = Double.parseDouble(JOptionPane.showInputDialog("Digite a taxa a ser aplicada no valor final:"));
        novoProduto.calculaValorVenda(taxaCalculo);

        //Para definir duas casas decimais na saida do numero ao imprimir
        DecimalFormat numero = new DecimalFormat("0.00");

        JOptionPane.showMessageDialog(null, "O produto " +novoProduto.getNome() + " da marca "
                                      +novoProduto.getMarca() +" tem preço final de: " +numero.format(novoProduto.getValorVenda()));
    }

}
```



Adaptando o programa principal para agora usar os métodos que acessam os atributos encapsulados

Encapsulamento – Outros exemplos

```
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
public class ProgramaEstoque {

    public static void main (String [] args){

        Produto novoProduto = new Produto();

        novoProduto.setNome("Escova dental");
        novoProduto.setCodigo("011");
        novoProduto.setMarca("Oral B");
        novoProduto.setValorCusto(1.90);

        double taxaCalculo = Double.parseDouble(JOptionPane.showInputDialog("Digite a taxa a ser aplicada no valor final:"));
        novoProduto.calculaValorVenda(taxaCalculo);

        //Para definir duas casas decimais na saida do numero ao imprimir
        DecimalFormat numero = new DecimalFormat("0.00");

        JOptionPane.showMessageDialog(null, "O produto " +novoProduto.getNome() + " da marca "
                                    +novoProduto.getMarca() +" tem preço final de: " +numero.format(novoProduto.getValorVenda()));

        novoProduto.valorVenda;

    }

}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        novoProduto cannot be resolved to a type
        Syntax error, insert "VariableDeclarators" to complete LocalVariableDeclaration

        at ProgramaEstoque.main(ProgramaEstoque.java:23)
```

```
Multiple markers at this line
  - Syntax error, insert "VariableDeclarators" to complete LocalVariableDeclaration
  - novoProduto cannot be resolved to a type
```

E se forçarmos a escrever o valor do preço final sem ser pelo método calculaValorVenda ou acessar qualquer atributo diretamente, veja o que acontece.

HERANÇA

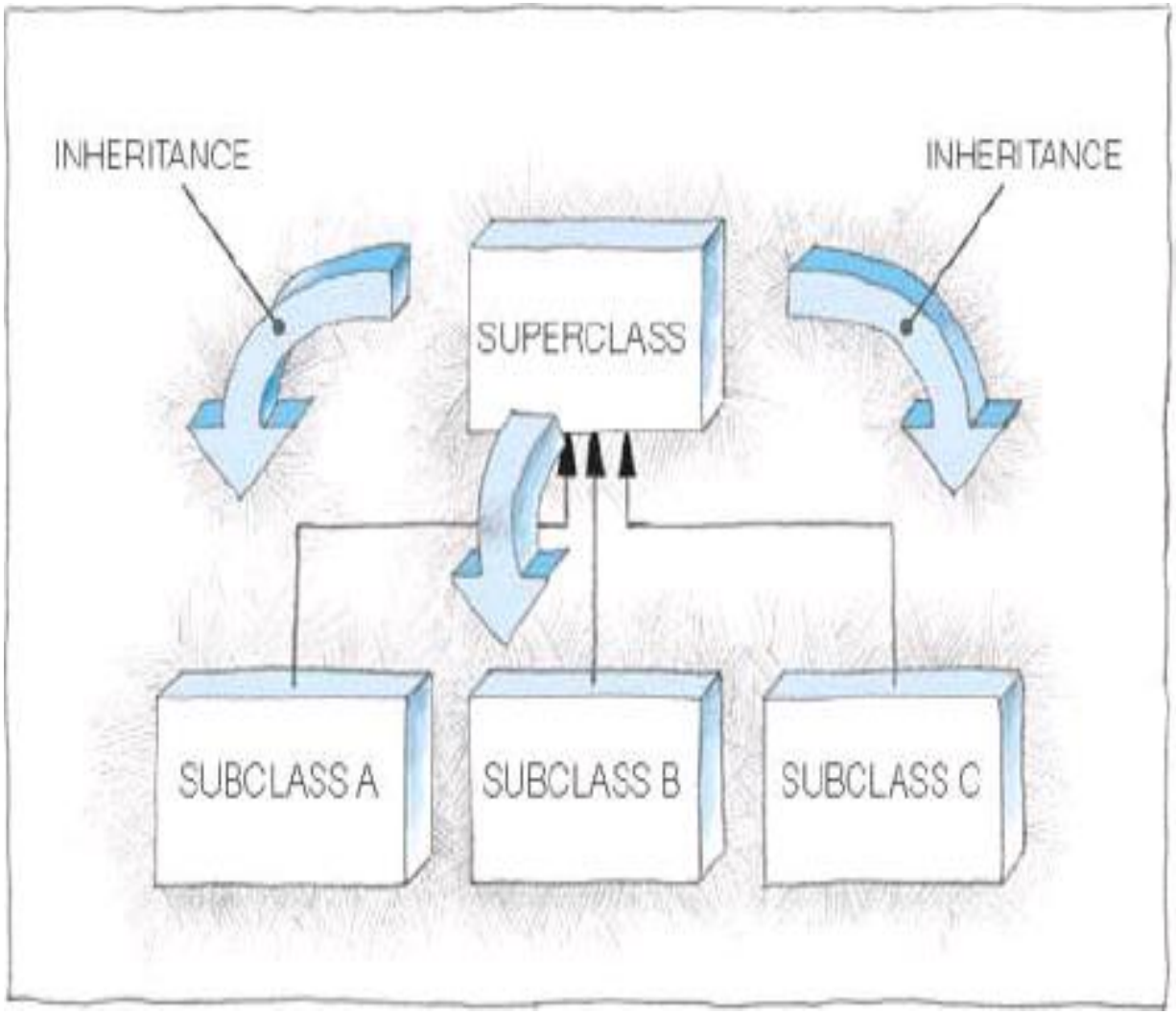
Herança

Herança (ou generalização) é o mecanismo pelo qual uma classe, a subclasse, **pode estender outra classe**, a superclasse, aproveitando seus comportamentos e estados possíveis (os membros da classe). Essa relação é normalmente chamada de relação "é um".

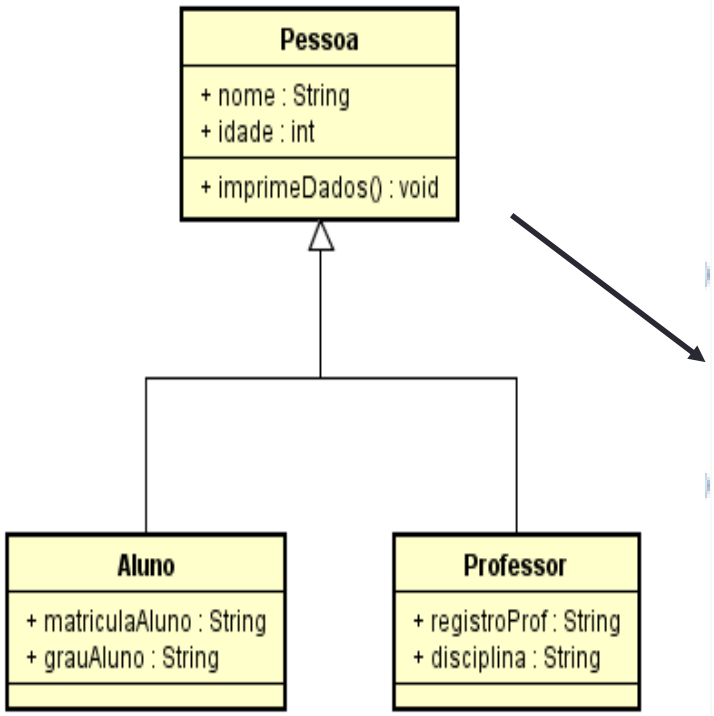
Assim, diz-se que as subclasses herdam a interface da superclasse, logo todas as subclasses de uma dada classe podem responder a qualquer mensagem que possa ser tratada pela superclasse (Polimorfismo), ou seja, um objeto pode substituir (em tempo de execução) um objeto da sua superclasse

Apesar do que pode parece a princípio, o ponto forte da herança não é a reutilização de código, já que a subclasse “herda” toda a implementação da superclasse, mas sim o polimorfismo.

Herança


INHERITANCE
INHERITANCE
SUPERCLASS
SUBCLASS A
SUBCLASS B
SUBCLASS C

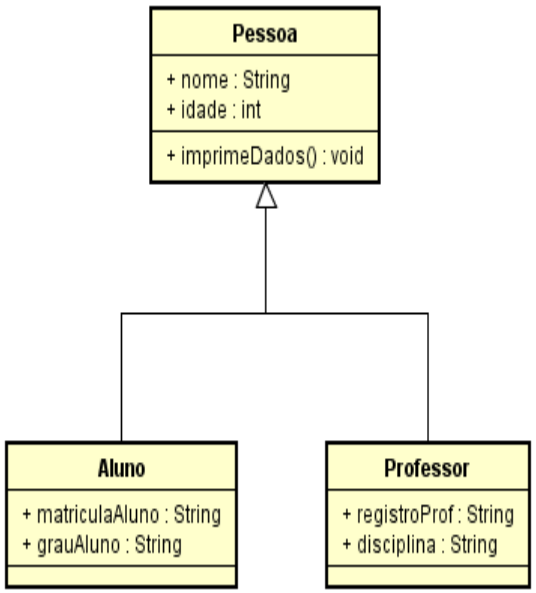
Herança




```
import javax.swing.JOptionPane;
public class Pessoa {

    String nome;
    int idade;

    Pessoa (String nome, int idade){
        this.nome = nome;
        this.idade = idade;
    }
    void imprimeDados(){
        JOptionPane.showMessageDialog(null, "Nome Aluno: " +this.nome
                                            +"Idade: " +this.idade);
    }

}
```

Herança

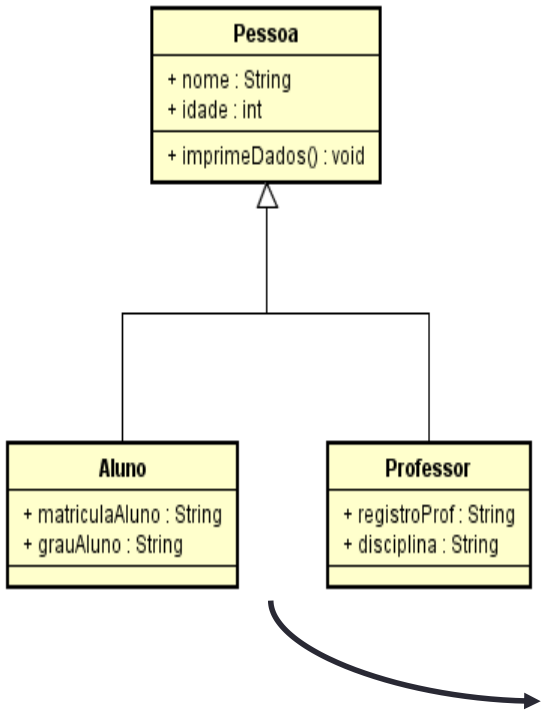
A palavra chave extends informa de qual superclasse provem a herança




```
import javax.swing.JOptionPane;
public class Aluno extends Pessoa{

    String matriculaAluno;
    String grauAluno;

    Aluno(String nome, int idade, String matricula, String grau){
        super(nome, idade);
        this.matriculaAluno = matricula;
        this.grauAluno = grau;
    }
    void imprimeDados(){
        JOptionPane.showMessageDialog(null, "Nome Aluno: " +this.nome +"\n"
                                            +"Idade: " +this.idade +"\n"
                                            +"Matricula: " +this.matriculaAluno +"\n"
                                            +"Grau: " +this.grauAluno);

        }

}
```

Herança

A classe Professor herda da classe Pessoa




```
import javax.swing.JOptionPane;
public class Professor extends Pessoa{

    String registroProf;
    String disciplina;

    Professor(String nome, int idade, String registro, String disciplina){
        super(nome, idade);
        this.registroProf = registro;
        this.disciplina = disciplina;
    }
    void imprimeDados(){
        JOptionPane.showMessageDialog(null, "Nome Professor: " +this.nome +"\n"
                                            +"Idade: " +this.idade +"\n"
                                            +"Registro: " +this.registroProf +"\n"
                                            +"Disciplina: " +this.disciplina);

        }

}
```

Herança

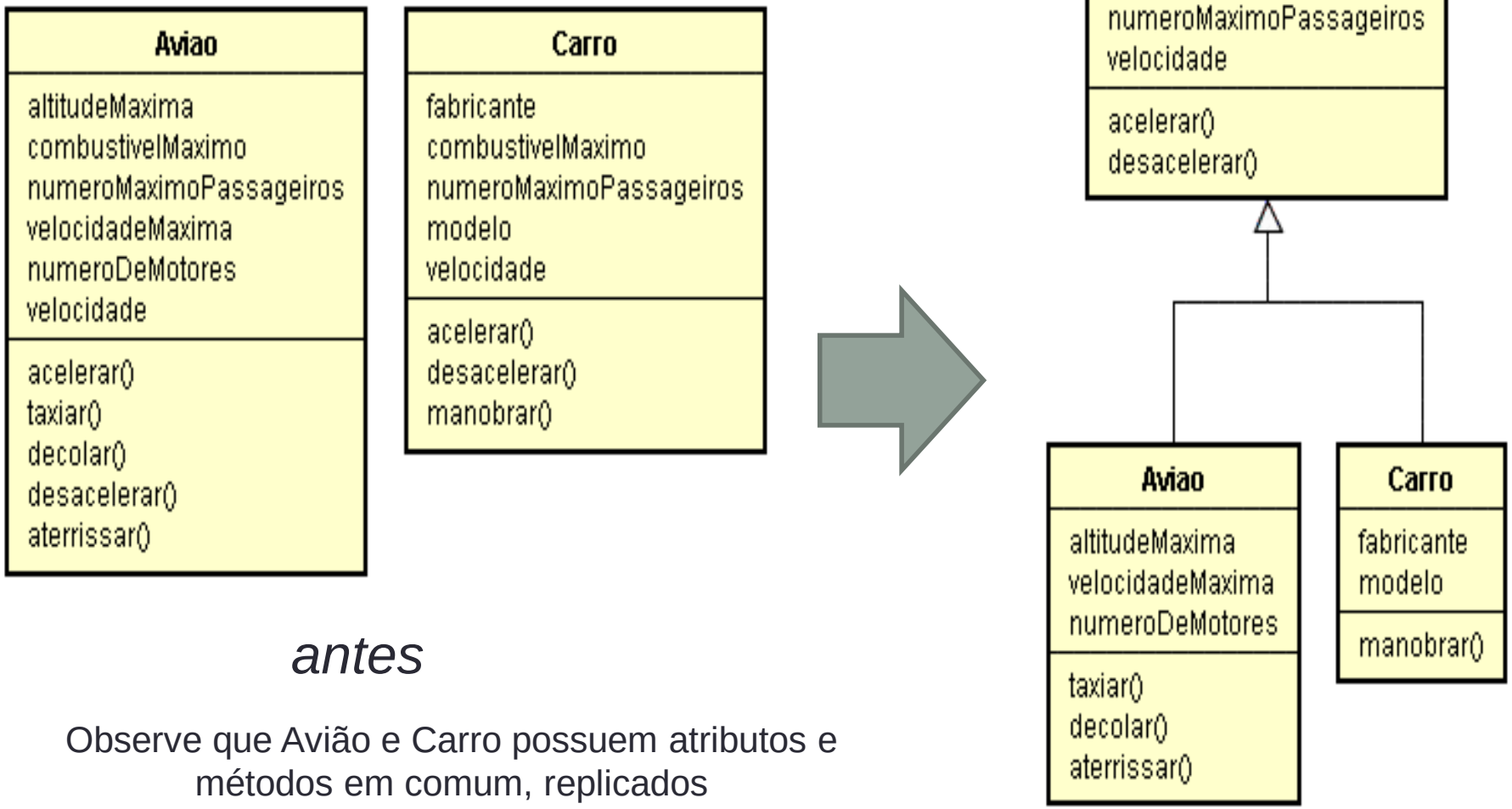
Executando um programa com a classe Aluno

```
public class ProgramaTesteAluno {

    public static void main (String [] args){

        Aluno novoAluno = new Aluno("Marcos Paulo", 18, "015B", "Ensino Médio");
        novoAluno.imprimeDados();
    }

}
```

Executando um programa com a classe Professor

```
public class ProgramaTesteProfessor {

    public static void main (String [] args){

        Professor novoProfessor = new Professor("José Rodrigues", 40, "3328", "Português");
        novoProfessor.imprimeDados();
    }
}
```

Herança



antes

Observe que Avião e Carro possuem atributos e métodos em comum, replicados

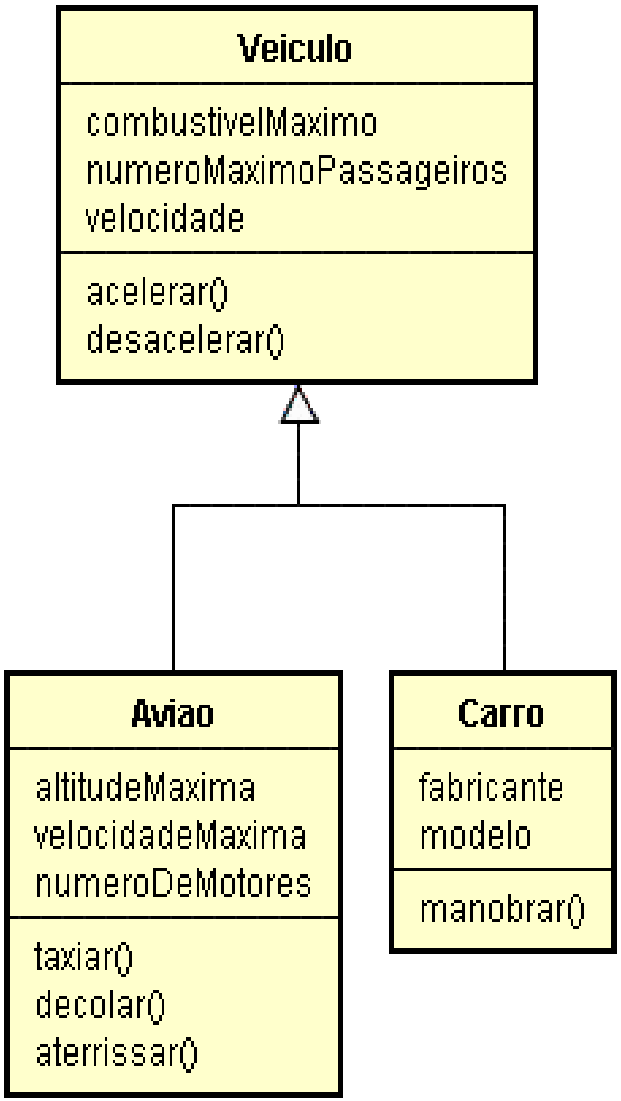
depois

Vamos praticar?

Escreva o código no exemplo das classes Pessoa, Professor e Aluno.

Execute e veja o que acontece.

Herança



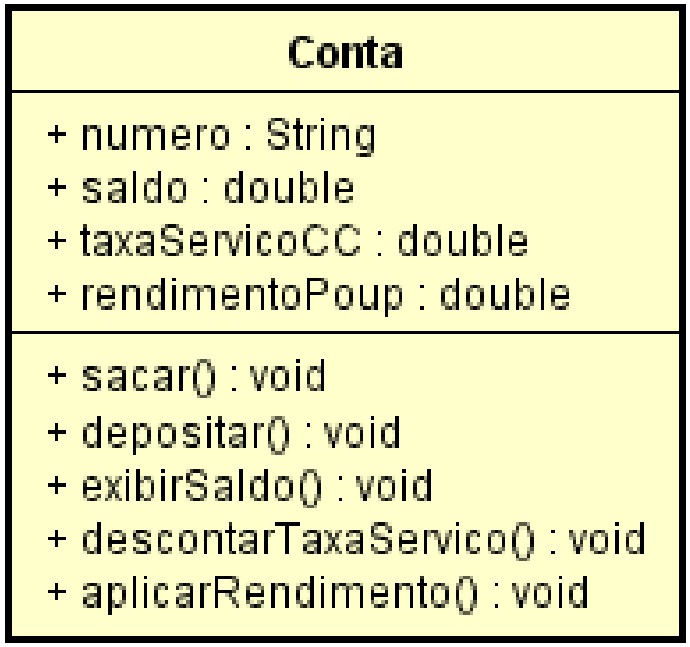
Vamos praticar 1?

Desenvolva o código em Java aplicando o conceito de Herança com base nas informações do diagrama ao lado.

Para os métodos das classes, elabore a ação que você preferir: imprimir uma mensagem informando que o veículo está taxiando, alterar o valor do atributo velocidade e mostrar na tela, etc.

Enfim, você é o programador e deve começar a pensar em como desenvolver as funções desempenhadas nos métodos.

Herança



Vamos praticar 2?

A classe conta ao lado possui alguns atributos e métodos que parecem estar a mais ou pertencerem a classes mais específicas.

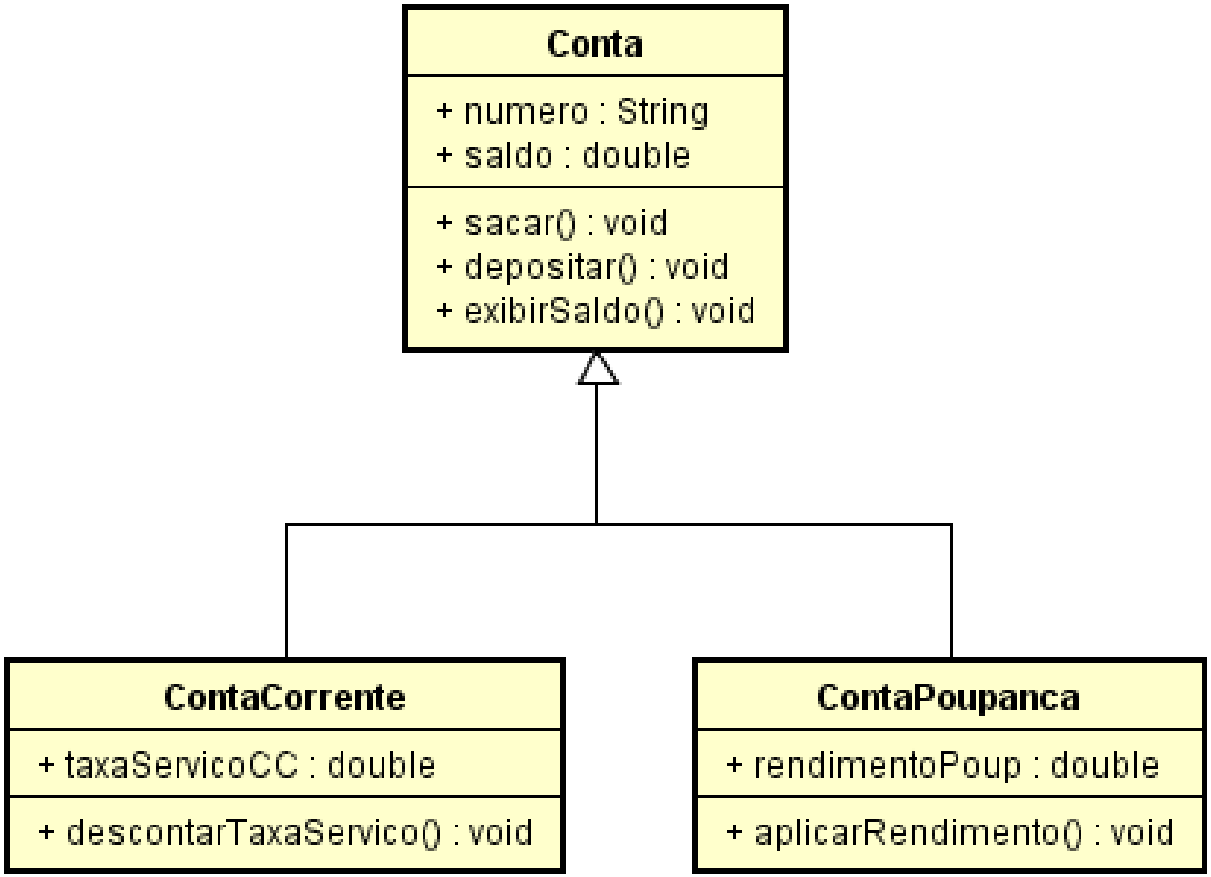
Aplique o conceito de Herança gerando classes mais especializadas, deixando os atributos e métodos comuns a todos na superclasse (ou classe mãe).

Desenvolva o código Java dessas novas classes. Não é necessário fazer o diagrama numa ferramenta, mas é recomendável que faça pelo menos um esboço, mesmo que seja no papel.

No próximo slide há um diagrama que é uma sugestão de resposta. Mas tente fazer o exercício sem consultá-lo.

Herança

Vamos praticar 2?



Sugestão de diagrama resposta ao slide anterior

KEEP CALM AND HAVE A BREAK