Relembrando…

1) Transforme os dois ArrayLists de alunos e notas em um mapa (HashMap).

```
13    public class Disciplina implements Comparable {
14
15        private HashMap<Aluno, Float> alunos;
16        private Professor professor;
17        private String nome;
18        private int ano;
```

```
26        public Disciplina() {
27            alunos = new HashMap();
28        }
```

```
42        public boolean registrarAluno(String nome, Curso curso,
43                int anoIngresso, long matricula) {
44            return alunos.put(
45                    new Aluno(nome, curso, anoIngresso, matricula),
46                    0f) != null;
47        }
```

Relembrando…

1) Transforme os dois ArrayLists de alunos e notas em um mapa (HashMap).

```
53    public boolean removerAluno(String nome) {
54        for (Iterator<Aluno> iterator = alunos.keySet().iterator();
55                iterator.hasNext();) {
56            Aluno aluno = iterator.next();
57
58            if (aluno.getNome().equals(nome)) {
59                iterator.remove();
60                return true;
61            }
62        }
63        return false;
64    }
```

Relembrando…

1) Transforme os dois ArrayLists de alunos e notas em um mapa (HashMap).

```
102     public Disciplina(Professor professor, String nome, int ano) {
103         this.professor = professor;
104         this.nome = nome;
105         this.ano = ano;
106         this.alunos = new HashMap();
107     }
```

```
 94     public HashMap<Aluno, Float> getAlunos() {
 95         return alunos;
 96     }
 97
 98     public Collection<Float> getNotas() {
 99         return alunos.values();
100     }
```

Relembrando…

1) Transforme os dois ArrayLists de alunos e notas em um mapa (HashMap).

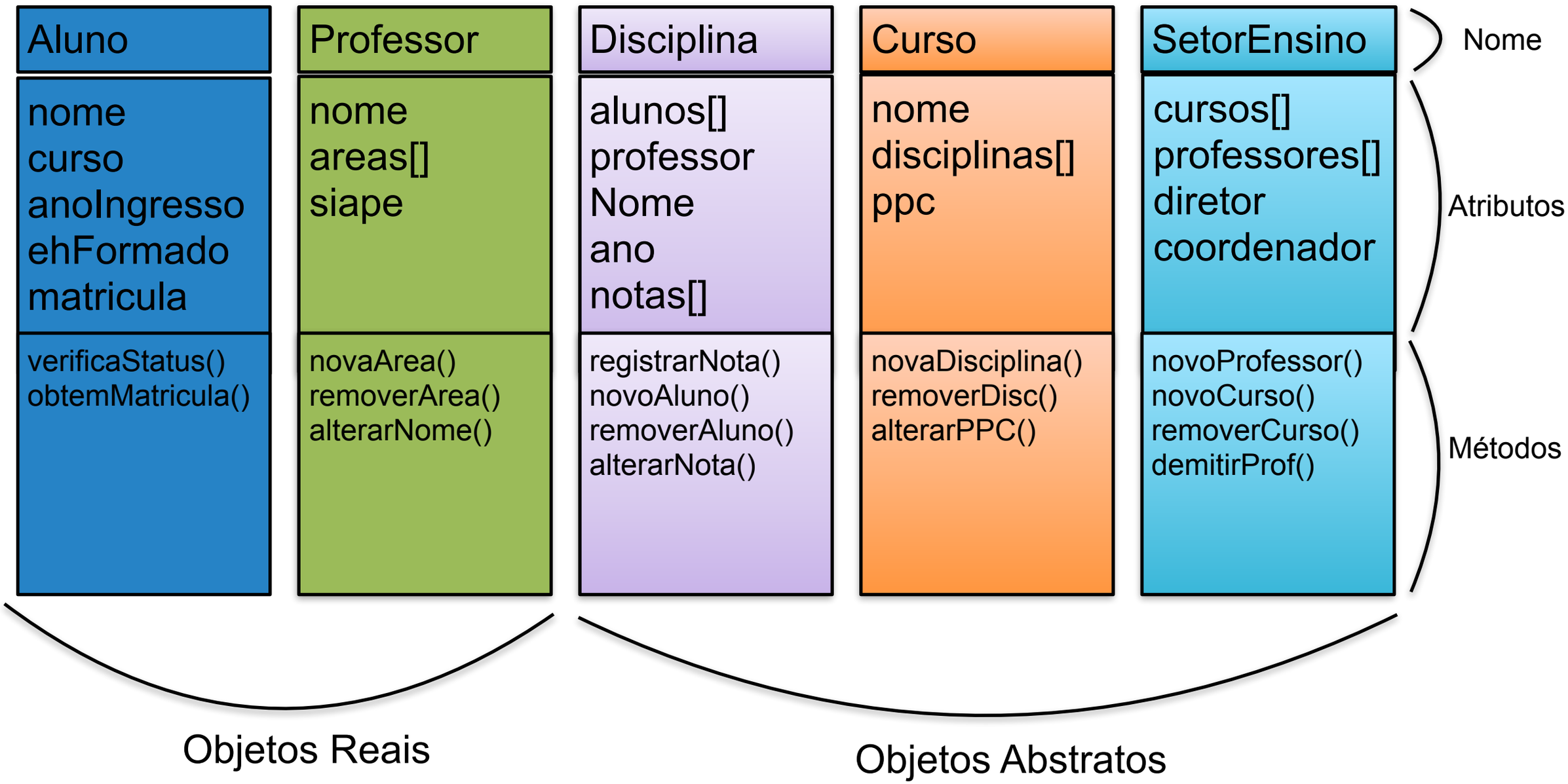
```
131         @Override
            public String toString() {
133             String notas_alunos = "";
134
135             for (Map.Entry<Aluno, Float> entry : alunos.entrySet()) {
136                 Aluno aluno = entry.getKey();
137                 Float nota = entry.getValue();
138
139                 if (aluno != null) {
140                     notas_alunos += aluno.toString() + " Nota: " + not
141                 }
142             }
143             return "\n professor: " + professor
144                     + "\n nome: " + nome
145                     + "\n ano: " + ano
146                     + "\n Notas: " + notas_alunos;
147         }
```

Herança de classes

Programação Orientada a Objetos

Rafael Vieira Coelho

Sistema Acadêmico Finalizado!



O que é herança de classes?

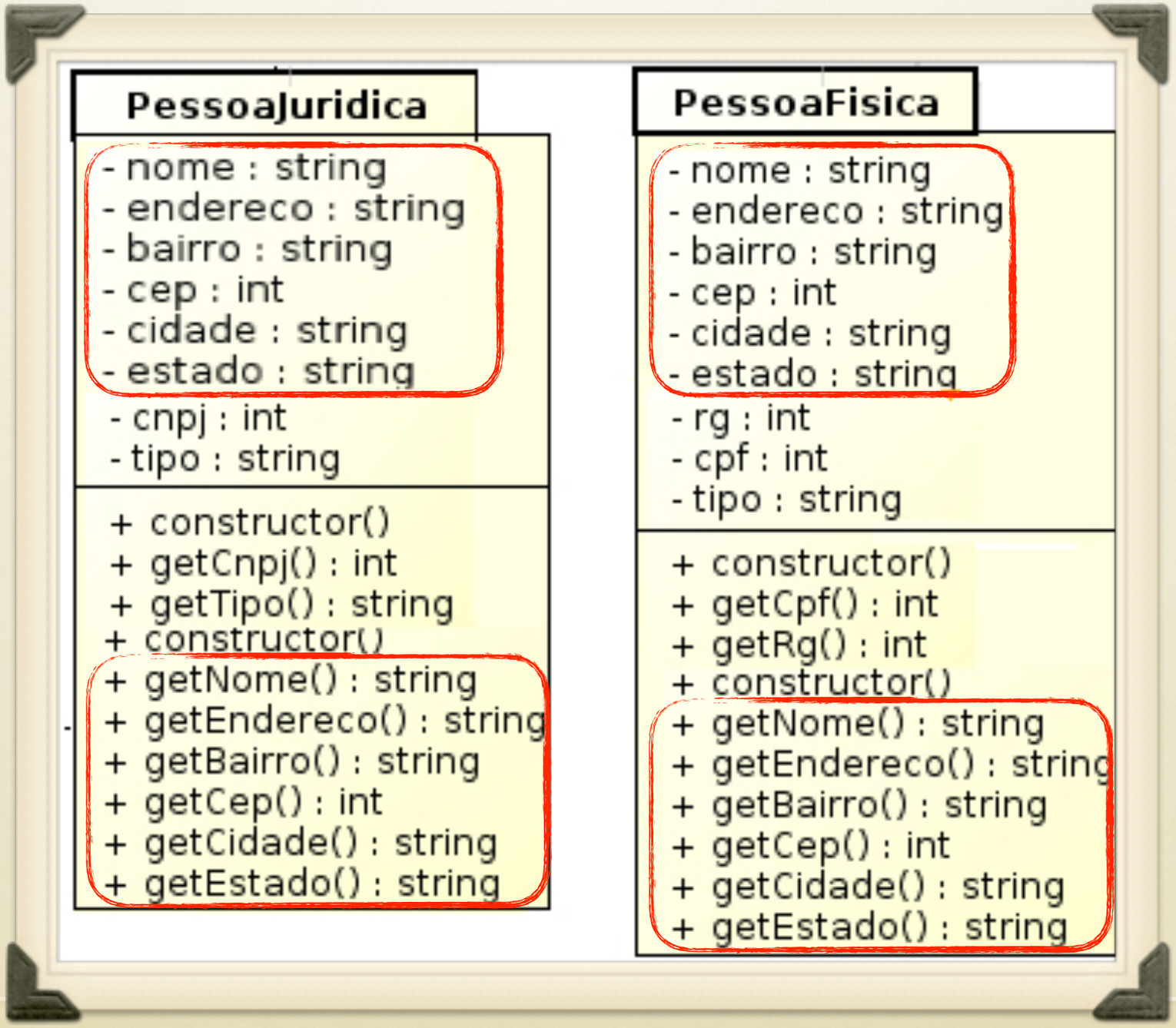
- **Herança** é um princípio de orientação a objetos, que permite que **classes** compartilhem atributos e métodos.

- Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.

- O conceito de **herança** de várias **classes** é conhecido como **herança** múltipla, mas não é suportado em Java.

Por que usar herança?

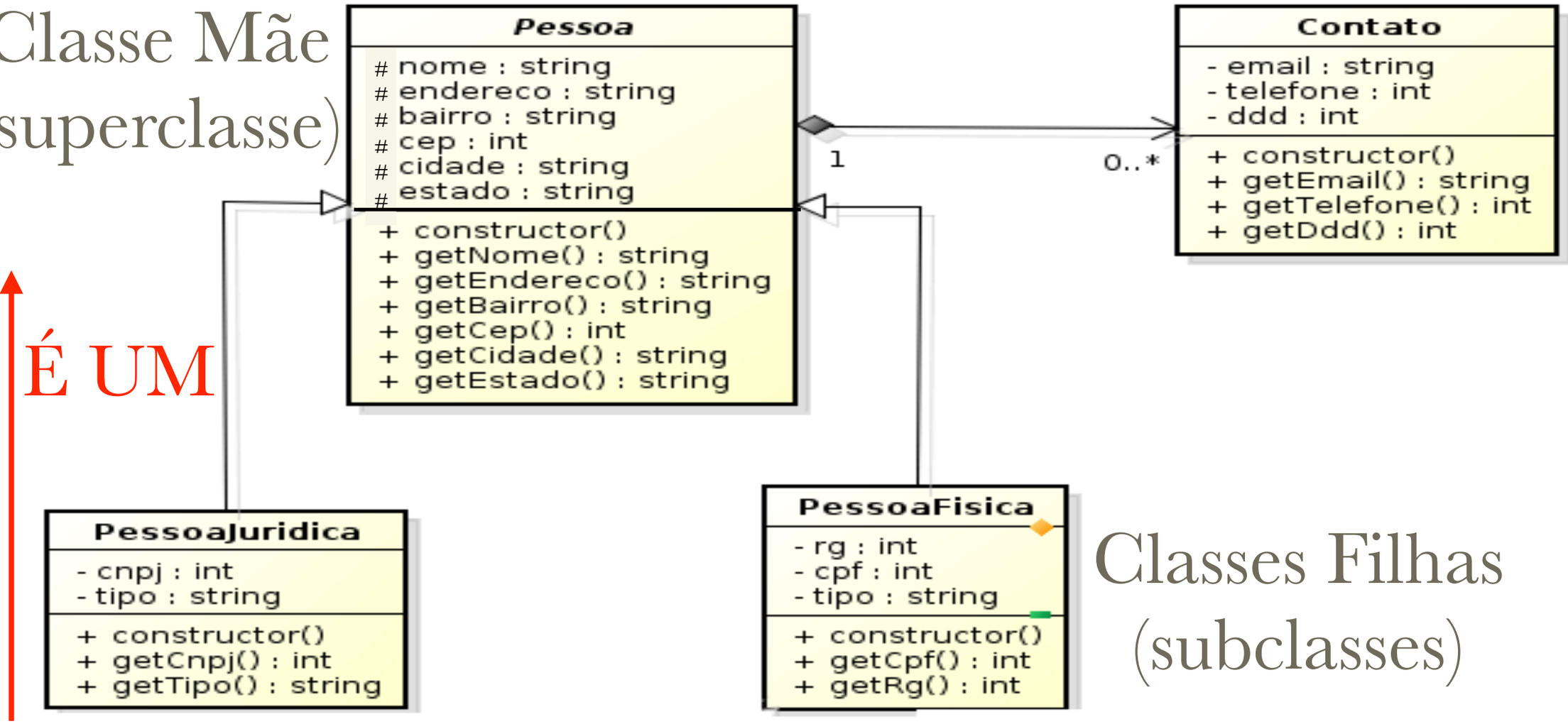
- O que temos em comum entre as 2 classes?



Implementaremos

Classe Mãe (superclasse)

É UM



Classes Filhas (subclasses)

Classe contato

- Cada pessoa terá uma lista de contatos.

```
7   public class Contato {
8       private String email;
9       private int telefone, ddd;
10
11      public String getEmail() {
12          return email;
13      }
14
15      public void setEmail(String email) {
16          this.email = email;
17      }
18
19      public int getTelefone() {
20          return telefone;
21      }
22
23      public void setTelefone(int telefone) {
24          this.telefone = telefone;
25      }
26
27      public int getDdd() {
28          return ddd;
29      }
30
31      public void setDdd(int ddd) {
32          this.ddd = ddd;
33      }
34
35      public Contato(String email, int telefone, int ddd) {
36          this.email = email;
37          this.telefone = telefone;
38          this.ddd = ddd;
39      }
40
41
42  }
43
```

Classe pessoa (superclasse)

- Atributos protegidos (protected) devido à herança.
- Métodos get/set

```
16      protected ArrayList<Contato> contatos;
17      protected String nome, endereco, bairro, cidade, estado;
18      protected int cep;
19
20      public ArrayList<Contato> getContatos() {
21          return contatos;
22      }
23
24      public void setContatos(ArrayList<Contato> contatos) {
25          this.contatos = contatos;
26      }
27
28      public String getNome() {
29          return nome;
30      }
31
32      public void setNome(String nome) {
33          this.nome = nome;
34      }
35
36      public String getEndereco() {
37          return endereco;
38      }
39
40      public void setEndereco(String endereco) {
41          this.endereco = endereco;
42      }
```

Classe pessoa

- Contrutor e Métodos Auxiliares

```
    public Pessoa(String nome, String endereco, String bairro, String cidade,
77          String estado, int cep) {
78      this.nome = nome;
79      this.endereco = endereco;
80      this.bairro = bairro;
81      this.cidade = cidade;
82      this.estado = estado;
83      this.cep = cep;
84      this.contatos = new ArrayList();
85  }
86
87  public boolean adicionaContato(Contato c) {
88      return this.contatos.add(c);
89  }
90
91  public boolean removeContato(Contato c) {
92      return this.contatos.remove(c);
93  }
94
95  public boolean existeContato(Contato c) {
96      return this.contatos.contains(c);
97  }
98
99  public void apagaContatos() {
100     this.contatos.clear();
101 }
```

Classe PessoaFisica (subclasse)

- Extende a classe Pessoa (extends)
- Atributos privados (private)
- Métodos get/set
- O construtor deve chamar o construtor da superclasse (super).

```
7   public class PessoaFisica extends Pessoa {
8       private int rg, cpf;
9       private String tipo;
10
11      public int getRg() {
12          return rg;
13      }
14
15      public void setRg(int rg) {
16          this.rg = rg;
17      }
18
19      public int getCpf() {
20          return cpf;
21      }
22
23      public void setCpf(int cpf) {
24          this.cpf = cpf;
25      }
26
27      public String getTipo() {
28          return tipo;
29      }
30
31      public void setTipo(String tipo) {
32          this.tipo = tipo;
33      }
34
35      public PessoaFisica(int rg, int cpf, String tipo, String nome,
36              String endereco, String bairro, String cidade,
37              String estado, int cep) {
38          super(nome, endereco, bairro, cidade, estado, cep);
39          this.rg = rg;
40          this.cpf = cpf;
41          this.tipo = tipo;
42      }
43  }
```

Classe PessoaJuridica (subclasse)

```
public class PessoaJuridica extends Pessoa {
    private int cnpj;
    private String tipo;

    public int getCnpj() {
        return cnpj;
    }

    public void setCnpj(int cnpj) {
        this.cnpj = cnpj;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public PessoaJuridica(int cnpj, String tipo, String nome, String endereco,
            String bairro, String cidade, String estado, int cep) {
        super(nome, endereco, bairro, cidade, estado, cep);
        this.cnpj = cnpj;
        this.tipo = tipo;
    }
}
```

Dúvidas?

1. Posso **reescrever um método** na classe filha que já existe na classe mãe?
2. Posso chamar qualquer construtor que esteja definido na classe mãe no construtor da classe filha através da palavra reservada **super**?
3. Como acesso os atributos e métodos da superclasse, estando na classe filha a partir da palavra reservada **super**?

Vamos para o IntelliJ para demonstrar!!

Tarefas

1. Implementar um programa principal no qual seja possível cadastrar, remover, atualizar e pesquisar por pessoas jurídicas e físicas.

2. Reutilize os métodos criados para permanência de dados (uso de arquivos). Desta forma, as informações cadastradas no programa não serão perdidas quando o mesmo for fechado.