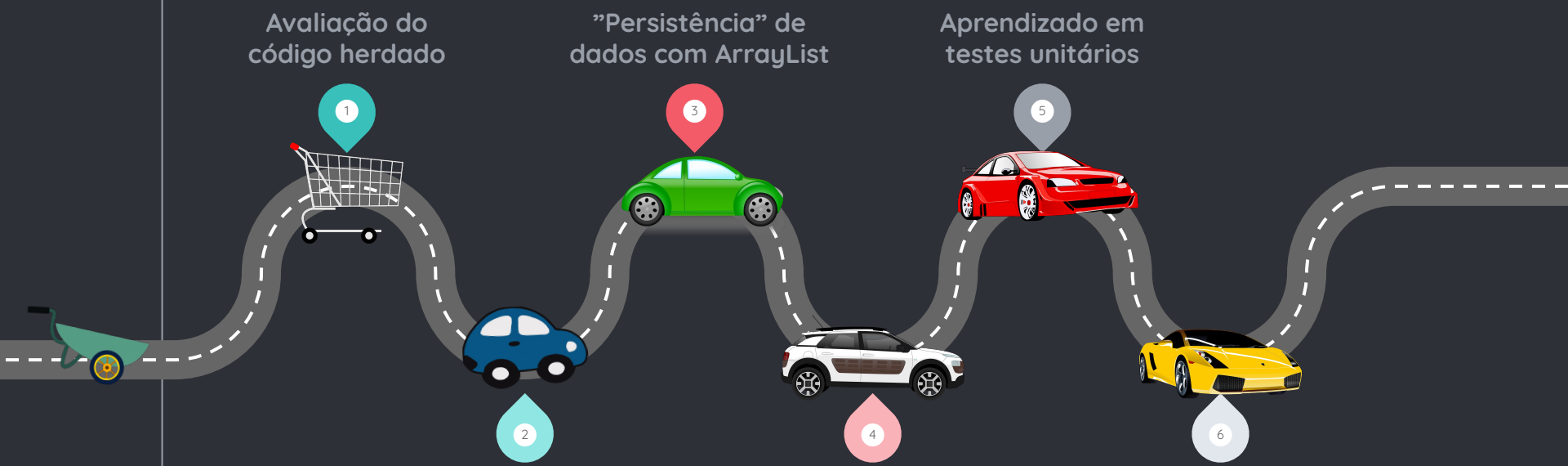
ProWay


Senior

# GRUPO 1 - CADASTRO

@author Lorran Pereira dos Santos
@author Samuel Levi
@author Sarah Neuburger Brito
@author Thiago Luiz Barbieri
@author Vitor Nathan Gonçalves

# Etapas da Sprint



# Sprint Review

O que foi feito?

- Redefinição dos requisitos;
- Criação das classes para POO;
- Criação dos testes unitários para os construtores das classes;
- Criação dos métodos nas classes;
- Implantação de validações (CEP, CPF, CNPJ, Telefone e E-mail);
- Testes unitários dos métodos;
- Diagrama UML;
- Documentação.

- O que não foi feito?
  - Validação das datas (nascimento, admissão e início contrato).
- O que foi acrescentado?
  - Validação das datas.
- O que foi retirado?
  - Teste unitário dos métodos de listagem.

# 1 Classes e Métodos

# Estrutura do Código

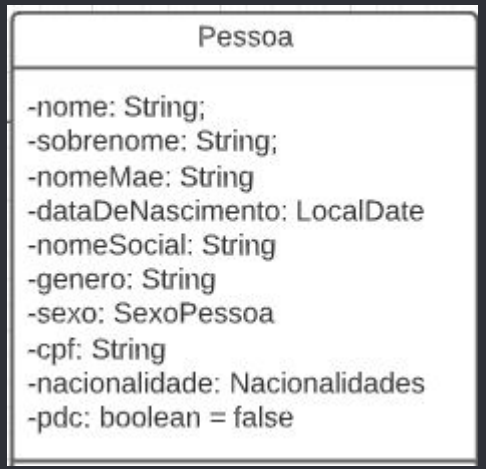
| Classe mãe | Classes filhas | Classes de apoio | Classes Enum |
| --- | --- | --- | --- |
| Pessoa | Colaborador | Endereco | TiposExame |
| | Dependente | Empresa | Cidades |
| | PrestadorServico | Contatos | UnidadesFederativas |
| | | ExameMedico | Paises |
| | | Documentos | TipoDependentes |
| | | | Nacionalidades |
| | | | SexoPessoa |

# Classes Enum

```
public enum UnidadesFederativas {

    /**
     * Classe de apoio para ser utilizada em Pessoa, Endereco e Colaborador para
     * setar informações de local de nascimento e endereço.
     * @author Lorran Pereira dos Santos, Samuel Levi, Sarah Neuburger Brito,
     * Thiago Luiz Barbieri e Vitor Nathan Gonçalves.
     */
    AC, AL, AP, AM, BA, CE, DF, ES, GO, MA, MT, MS, MG, PA, PB, PR, PE, PI, RJ,
RN, RS, RO, RR, SC, SP, SE, TO
}
```

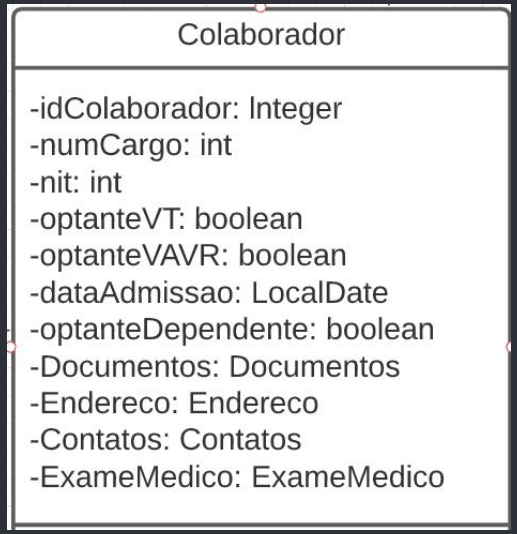
# Pessoa - Construtor

```
public Pessoa(String nome, String sobrenome, LocalDate dataDeNascimento,
String nomeSocial, String genero, SexoPessoa sexo, String nomeDaMae,
String cpf, Nacionalidades nacionalidade, boolean pcd) {
    this.nome = nome;
    this.sobrenome = sobrenome;
    this.dataDeNascimento = dataDeNascimento;
    this.nomeSocial = nomeSocial;
    this.genero = genero;
    this.sexo = sexo;
    this.nomeDaMae = nomeDaMae;
    this.cpf = cpf;
    this.nacionalidade = nacionalidade;
    this.pcd = pcd;
}
```



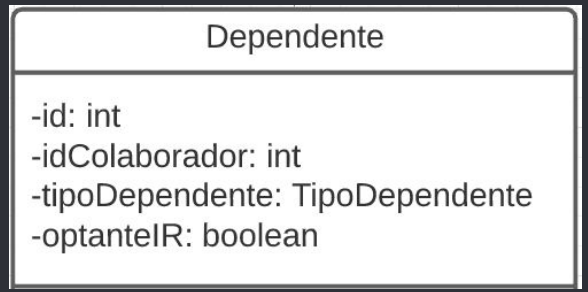
# Colaborador - Construtor

```
public Colaborador(String nome, String sobrenome, LocalDate dataDeNascimento, String nomeSocial, String genero,
        SexoPessoa sexo, String nomeDaMae, String cpf, Nacionalidades nacionalidade, boolean pcd,
        Integer idColaborador, Integer numCargo, Integer nit, boolean optanteVT, boolean optanteVAVR,
        LocalDate dataAdmissao, boolean optanteDependente, Cidades cidadeNascimento,
        UnidadesFederativas ufNascimento, Paises paisNascimento, Documentos documentos, Endereco endereco,
        Contatos contatos, ExameMedico exameMedico) {
    super(nome, sobrenome, dataDeNascimento, nomeSocial, genero, sexo, nomeDaMae, cpf, nacionalidade, pcd);
    this.idColaborador = idColaborador;
    this.numCargo = numCargo;
    this.nit = nit;
    this.optanteVT = optanteVT;
    this.optanteVAVR = optanteVAVR;
    this.dataAdmissao = dataAdmissao;
    this.optanteDependente = optanteDependente;
    this.cidadeNascimento = cidadeNascimento;
    this.UfNascimento = ufNascimento;
    this.paisNascimento = paisNascimento;
    this.documentos = documentos;
    this.endereco = endereco;
    this.contatos = contatos;
    this.exameMedico = exameMedico;
}
    return true;
}
```



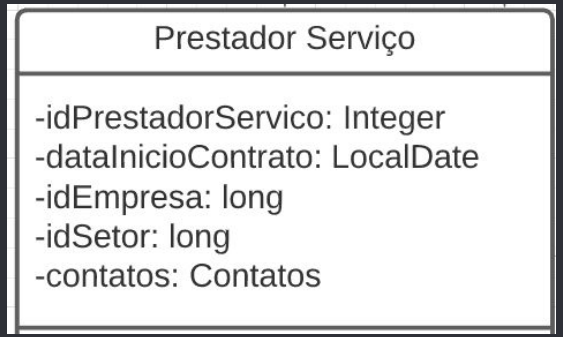
# Dependente - Construtor

```
public Dependente(String nome, String sobrenome, LocalDate
dataDeNascimento, String nomeSocial, String genero,SexoPessoa sexo,
String nomeDaMae, String cpf, Nacionalidades nacionalidade, boolean
pcd, Integer id,Integer idColaborador, TiposDependentes tipoDependente,
boolean optanteIR) {
        super(nome, sobrenome, dataDeNascimento, nomeSocial, genero,
sexo, nomeDaMae, cpf, nacionalidade, pcd);
        this.id = id;
        this.idColaborador = idColaborador;
        this.tipoDependente = tipoDependente;
        this.optanteIR = optanteIR;
}
```



# Prestador de Serviço - Construtor

```
public PrestadorServico(String nome, String sobrenome, LocalDate dataDeNascimento,
String nomeSocial, String genero, SexoPessoa sexo, String nomeDaMae, String cpf,
Nacionalidades nacionalidade, boolean pcd, Integer idPrestadorServico, LocalDate
dataInicioContrato, Integer idEmpresa, Integer idSetor, Contatos contatos) {
    super(nome, sobrenome, dataDeNascimento, nomeSocial, genero, sexo, nomeDaMae,
cpf, nacionalidade, pcd);
    this.idPrestadorServico = idPrestadorServico;
    this.dataInicioContrato = dataInicioContrato;
    this.idEmpresa = idEmpresa;
    this.idSetor = idSetor;
    this.contatos = contatos;
}
```



# Métodos CRUD - Create

```java
/**
 * Cadastrar prestador de serviço
 * Realiza cadastro de um prestador de serviço
 * @param ArrayList<PrestadorServico> listaPrestadorServico
 * @param PrestadorServico            prestador
 * @return void
 */
public static void cadastrarPrestadorServico(ArrayList<PrestadorServico> listaPrestadorServico,
PrestadorServico prestador) {
    listaPrestadorServico.add(prestador);
}
@Test
public void testCadastrarPrestadorServico() {
    PrestadorServico.cadastrarPrestadorServico(listaPrestadorServico, prestador1);
    PrestadorServico.cadastrarPrestadorServico(listaPrestadorServico, prestador2);
    assertTrue(listaPrestadorServico.indexOf(prestador1) == 0);
    assertTrue(listaPrestadorServico.indexOf(prestador2) == 1);
}
```

# Métodos CRUD - Read

```java
/**
 * Listar todos os exames médicos
 * Realiza a leitura de todos os exames médicos cadastrados e apresenta na tela.
 * @param ExameMedico exame
 * @param ArrayList   listaExames
 * @return ArrayList
 */
public static ArrayList<ExameMedico> listarTodosExames(ArrayList<ExameMedico> listaExames) {
    for (ExameMedico exame2 : listaExames) {
        System.out.println(exame2);
    }
    return listaExames;
}
@Test @Ignore
public void testListarTodosExames() {
      ExameMedico.cadastrarExameMedico(listaExames, exame1);
      ExameMedico.cadastrarExameMedico(listaExames, exame2);
      ExameMedico.listarTodosExames(listaExames);
      assertEquals(tipoExame1, ExameMedico.listarTodosExames(listaExames).get(0).getTipoExame());
      assertEquals(dataExame1, ExameMedico.listarTodosExames(listaExames).get(0).getDataExame());
      assertEquals(apto1, ExameMedico.listarTodosExames(listaExames).get(0).isApto());
      assertEquals(tipoExame2, ExameMedico.listarTodosExames(listaExames).get(1).getTipoExame());
      assertEquals(dataExame2, ExameMedico.listarTodosExames(listaExames).get(1).getDataExame());
      assertEquals(apto2, ExameMedico.listarTodosExames(listaExames).get(1).isApto());
    }
```

# Métodos CRUD - Update

```java
/**
 * Atualizar uma empresa.
 * Substitui o objeto "Empresa" antigo, pelo novo objeto informado no parâmetro.
 * @param ArrayList Recebe uma instância de ArrayList.
 * @param Empresa   Recebe um objeto da empresa nova
 * @param Empresa   Recebe o objeto referente à empresa antiga
 */
public static void atualizarEmpresa(ArrayList<Empresa> listaEmpresas, Empresa empresaNova, Empresa empresaAntiga)
{
    int indice = listaEmpresas.indexOf(empresaAntiga);
    listaEmpresas.set(indice, empresaNova);
}

@Test
public void testeAtualizarEmpresa() {
    ArrayList<Empresa> listaEmpresas = new ArrayList<Empresa>();
    listaEmpresas.add(empresa);
    Empresa.atualizarEmpresa(listaEmpresas, novaEmpresa, empresa);
    assertEquals(listaEmpresas.get(0), novaEmpresa);
    assertFalse(listaEmpresas.get(0) == empresa);
}
```

# Métodos CRUD - Delete

```java
/**
 * Remove um contato da lista
 * Este método remove um contato da lista e faz uma verificação se o objeto contato
 * informado estiver na lista, caso não esteja, retorna uma String de aviso.
 * @param ArrayList<Contatos> listaContatos
 * @param Contatos contato
 * @return void
 */
public static void deletarContatoPassandoUmContato(ArrayList<Contatos> listaContatos, Contatos contato) {
    listaContatos.remove(contato);
}
@Test
public void testDeletarUmContatoPassandoUmContato() {
    Contatos contato1 = new Contatos("(47) 99999-9999", "(47) 12345-6789", "456@teste.com.br");
    Contatos.cadastrarContato(listaContatos, contato1);
    Contatos contato2 = new Contatos("(47) 99999-9999", "(47) 12345-6789", "123@teste.com.br");
    Contatos.cadastrarContato(listaContatos, contato2);
    Contatos.removerUmContatoPassandoUmContato(listaContatos, contato1);
    assertEquals("47999999999", listaContatos.get(0).getTelefonePrincipal());
    assertEquals("47123456789", listaContatos.get(0).getTelefoneSecundario());
    assertEquals("123@teste.com.br", listaContatos.get(0).getEmail());
}
```

# Métodos de Validação

```
public static boolean validarCPF(String CPF) {
    String CPFFormatado = Pessoa.formatarCPF(CPF);
    if (CPF.length() == 11) {

        int soma = 0;
        int mult = 10;
        for (int i = 0; i < 9; i++) {
            soma += mult *
((CPFFormatado.charAt(i) - 48));
            mult--;
        }
        if (11 - soma % 11 ==
CPFFormatado.charAt(9) - 48) {
            boolean valido;
        } else if (11 - soma % 11 == 10 &&
CPFFormatado.charAt(9) - 48 == 0) {
            boolean valido;
        } else {
            return false;
        }
        soma = 0;
        mult = 11;
        for (int i = 0; i < 10; i++) {
            soma += mult * ((CPFFormatado.charAt(i) - 48));
            mult--;
        }
        if (11 - soma % 11 == CPFFormatado.charAt(10) - 48) {
            boolean valido;
        } else if (11 - soma % 11 == 10 &&
CPFFormatado.charAt(10) - 48 == 0) {
            boolean valido;
        } else {
            return false;
        }

    } else {
        return false;
    }
    return true;
}
```

# Métodos de Validação

```
public static String formatarCPF(String CPF) {
    String output = "";
    for (byte code : CPF.getBytes()) {
        if (code - 48 < 10 && code - 48 >= 0) {
            output += Character.toString((char) code);
        }
    }
    return output;
}
public static boolean validarEmail(String email) {
        if (!email.contentEquals("@")) {
            return false;
        }
        return true;
}
```

# Métodos de Validação

```
public static boolean validarTamanhoTel(String telefone) {
       if(telefone.length() != 11) {
            return false;
       }
       return true;
}

public static boolean validarCEP(String cep) {
        if(Endereco.formatarCEP(cep).length() != 8) {
            return false;
        }
        return true;
}
```

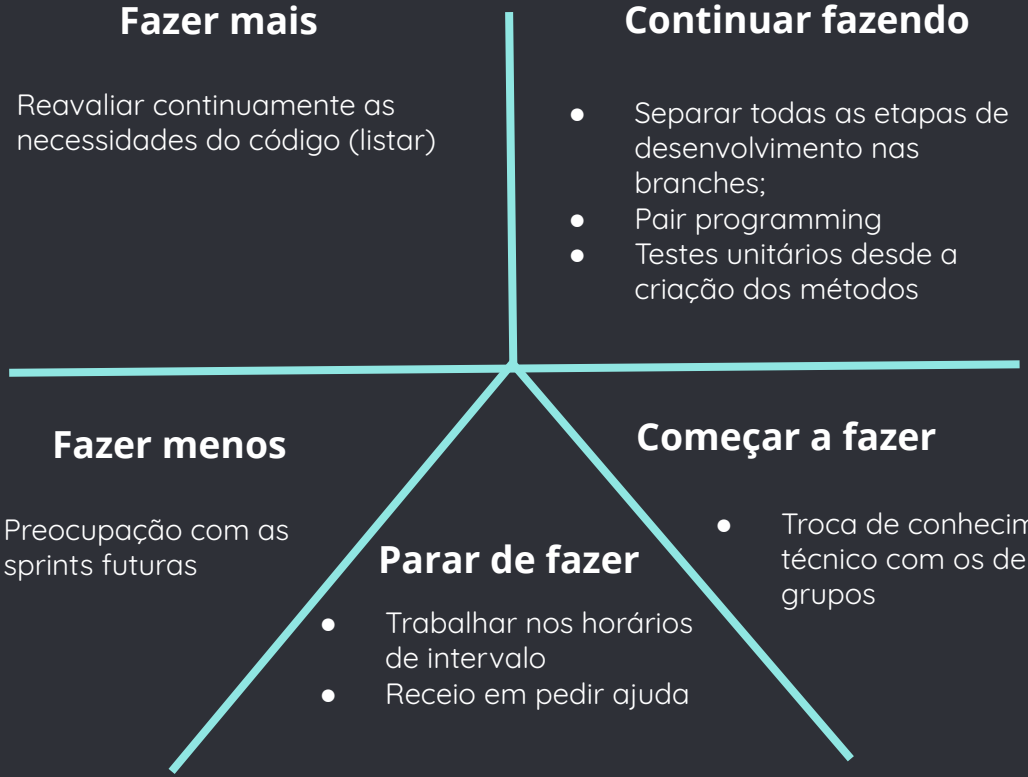
# Métodos de Validação

```
ppublic static boolean validaCNPJ(String cnpj) {

    String cnpjFormatado = Empresa.formataCNPJ(cnpj);
    if (cnpjFormatado.length() == 14) {
        String cnpjInvertido = "";
        for (int i = 13; i >= 0; i--) {
            cnpjInvertido += cnpjFormatado.charAt(i);
        }
        int mult = 2;
        int soma = 0;
        for (int i = 2; i < 14; i++) {
            soma += mult * (cnpjInvertido.charAt(i) - 48);
            if (mult == 9) {
                mult = 2;
            } else {
                mult++;
            }
        }
        if (cnpjInvertido.charAt(1) - 48 != 11 - (soma % 11)) {
            return false;
        }
        mult = 2;
        soma = 0;
        for (int i = 1; i < 14; i++) {
            soma += mult * (cnpjInvertido.charAt(i) - 48);
            if (mult == 9) {
                mult = 2;
            } else {
                mult++;
            }
        }
        if (cnpjInvertido.charAt(0) - 48 != 11 - (soma % 11)) {
            return false;
        }

    }
    return true;
}
```

# Sprint Retrospective

- O que deu certo?
  - Testes unitários;
  - Utilização do Git/Github;
  - Implementação das Classes e Métodos;
  - Aplicação do Pair Programming;
  - Alinhamento entre o grupo antes de iniciar o desenvolvimento.
  - Organização do Trello.
- O que deu errado?
  - Testes dos métodos de listar com ArrayList.
- O que eu aprendi?
  - Uso do Git/Github;
  - Uso do ArrayList;
  - Aplicação do Trello para comunicação interna;
  - POO.
- O que devemos fazer diferente na próxima Sprint?
  - Padronização de nomenclatura dos métodos.

# Sprint Retrospective



# Dúvidas?

## Agradecemos pela atenção!