PaceHub - Diagrama de Classes (Entrega 2)

Especificação UML para Enterprise Architect

Classes de Domínio com Atributos e Relacionamentos

```
@startuml PaceHub_ClassDiagram

abstract class Usuario {
   # cpf: String
   # nome: String
   # email: String
   # telefone: String
   # dataNascimento: Date
   # senha: String
   + validarCPF(): boolean
   + calcularIdade(): int
   + autenticar(): boolean
}

class Atleta {
   - numeroInscricoes: int
   + inscreverEvento(): boolean
   + cancelarInscricao(): boolean
   + consultarResultados(): List<Resultado>
   + preencherFichaMedica(): boolean
}

class Organizador {
   - cnpj: String
   - razaoSocial: String
   + criarEvento(): Evento
   + importarResultados(): boolean
   + gerarRelatorioEstatisticas(): RelatorioEstatisticas
}

class Evento {
   - id: String
   - nome: String
   - descricao: String
   - dataEvento: Date
   - horaInicio: Time
   - distancia: DistanciaEnum
```

```
    - tempoCorte: int
    - prazoInscricao: Date
    - prazoCancelamento: Date
    - localRealizacao: String
    - status: StatusEvento
    - maxParticipantes: int
    + validarIdadeMinima(): boolean
    + calcularEstatisticas(): RelatorioEstatisticas
    + alterarStatus(): void
}

class Inscricao {
    - id: String
    - dataInscricao: Date
    - status: StatusInscricao
    - kitEntregue: boolean
    - fichaPreenchida: boolean
    - termoAceito: boolean
    + confirmar(): boolean
    + cancelar(): boolean
    + entregarKit(): boolean
}

class FichaMedica {
    - id: String
    - dataPreenchimento: Date
    - problemaCardiaco: boolean
    - usaMedicacao: boolean
    - observacoes: String
    - aprovada: boolean
    + validar(): boolean
    + aprovar(): boolean
}

class TermoResponsabilidade {
    - id: String
    - dataAceite: Date
    - ipOrigemAceite: String
    - versaoTermo: String
    + registrarAceite(): boolean
}

class Resultado {
    - id: String
    - tempoFinal: Time
    - posicaoGeral: int
    - posicaoCategoria: int
    - categoria: CategoriaEnum
```

```
    - classificado: boolean
    + calcularPosicoes(): void
    + validarTempoCorte(): boolean
}

class RelatorioEstatisticas {
    - totalInscritos: int
    - totalHomens: int
    - totalMulheres: int
    - kitsEntregues: int
    - dataGeracao: Date
    + gerarEstatisticas(): void
    + exportar(): String
}

enum DistanciaEnum {
    KM_5(5, 14)
    KM_10(10, 16)
    KM_21(21, 18)
    KM_42(42, 20)
    --
    + distanciaKm: int
    + idadeMinima: int
    + validarIdade(): boolean
}

enum CategoriaEnum {
    GERAL_MASCULINO
    GERAL_FEMININO
    JUNIOR_MASCULINO
    JUNIOR_FEMININO
    ADULTO_MASCULINO
    ADULTO_FEMININO
    MASTER_MASCULINO
    MASTER_FEMININO
    PCD_ESPECIAL
}

enum StatusEvento {
    PLANEJAMENTO
    INSCRICOES_ABERTAS
    INSCRICOES_FECHADAS
    EM_ANDAMENTO
    FINALIZADO
    CANCELADO
}

enum StatusInscricao {
```

```
    PENDENTE
    CONFIRMADA
    CANCELADA
    KIT_ENTREGUE
}

' Relacionamentos
Usuario <|-- Atleta
Usuario <|-- Organizador

Organizador ||--o{ Evento : gerencia
Atleta ||--o{ Inscricao : realiza
Evento ||--o{ Inscricao : recebe
Inscricao ||--|| FichaMedica : possui
Inscricao ||--|| TermoResponsabilidade : assina
Atleta ||--o{ Resultado : obtem
Evento ||--o{ Resultado : gera
Evento ||--|| RelatorioEstatisticas : produz

@enduml
```

Detalhamento das Classes

1. Usuario (Classe Abstrata)

Estereótipo: `<<abstract>>`

- **Responsabilidade**: Representa características comuns de usuários do sistema
- **Atributos**:
  - `cpf`: String - Identificador único (RN11, RN12)
  - `nome`: String - Nome completo do usuário
  - `email`: String - E-mail para comunicação
  - `telefone`: String - Contato telefônico
  - `dataNascimento`: Date - Para cálculo de idade (RN04)
  - `senha`: String - Hash da senha (RNF05)

2. Atleta

Herda de: Usuario

- **Responsabilidade**: Gerenciar dados e ações específicas do atleta
- **Atributos adicionais**:
  - `numeroInscricoes`: int - Controle quantitativo
- **Relacionamentos**: 1:N com Inscricao, 1:N com Resultado

3. Organizador

Herda de: Usuario

- **Responsabilidade**: Gerenciar eventos e suas funcionalidades
- **Atributos adicionais**:
  - `cnpj`: String - Para organizadores pessoa jurídica
  - `razaoSocial`: String - Nome empresarial
- **Relacionamentos**: 1:N com Evento

4. Evento

- **Responsabilidade**: Centralizar informações do evento de corrida
- **Atributos críticos**:
  - `distancia`: DistanciaEnum - Determina idade mínima (RN01)
  - `tempoCorte`: int - Tempo máximo em horas (RN03)
  - `status`: StatusEvento - Controla fluxo do evento
- **Relacionamentos**: N:1 com Organizador, 1:N com Inscricao, 1:N com Resultado

5. Inscricao

- **Responsabilidade**: Controlar processo de inscrição do atleta
- **Atributos de controle**:
  - `kitEntregue`: boolean - Gerenciamento de kits (RF05)
  - `fichaPreenchida`: boolean - Validação RN05
  - `termoAceito`: boolean - Validação RN05
- **Relacionamentos**: N:1 com Atleta, N:1 com Evento, 1:1 com FichaMedica, 1:1 com TermoResponsabilidade

6. Resultado

- **Responsabilidade**: Armazenar e processar resultados da corrida
- **Atributos para classificação**:
  - `posicaoGeral`: int - Top 5 por gênero (RN06)
  - `posicaoCategoria`: int - Classificação por faixa etária (RN07)
  - `categoria`: CategoriaEnum - Categoria do atleta (RN04)
- **Relacionamentos**: N:1 com Atleta, N:1 com Evento

Enumerações Detalhadas

DistanciaEnum

- **Propósito**: Implementar RN01 (validação de idade mínima)
- **Valores**: Cada constante possui distância e idade mínima
- **Método**: `validarIdade(atleta)` para verificar RN01

CategoriaEnum

- **Propósito**: Implementar RN06 e RN07 (classificações)
- **Hierarquia**: Geral (top 5) → Por faixa etária → PCD

Validações e Regras Implementadas

RN01 - Idade Mínima por Distância

```
// No método Evento.validarIdadeMinima()
int idadeAtleta = atleta.calcularIdade(); // RN04
return this.distancia.validarIdade(idadeAtleta);
```

RN03 - Tempo de Corte

```
// No método Resultado.validarTempoCorte()
return this.tempoFinal <= evento.getTempoCorte();
```

RN04 - Cálculo de Idade

```
// No método Usuario.calcularIdade()
Calendar cal = Calendar.getInstance();
int anoEvento = evento.getDataEvento().getYear();
return anoEvento - this.dataNascimento.getYear();
```

RN05 - Ficha Médica e Termo

```
// No método Inscricao.confirmar()
return this.fichaPreenchida && this.termoAceito;
```

Multiplicidades dos Relacionamentos

- **Organizador** → **Evento**: 1:N (um organizador gerencia vários eventos)
- **Atleta** → **Inscricao**: 1:N (um atleta pode se inscrever em vários eventos)
- **Evento** → **Inscricao**: 1:N (um evento recebe várias inscrições)
- **Inscricao** → **FichaMedica**: 1:1 (cada inscrição tem uma ficha)
- **Inscricao** → **TermoResponsabilidade**: 1:1 (cada inscrição tem um termo)
- **Atleta** → **Resultado**: 1:N (um atleta pode ter vários resultados)
- **Evento** → **Resultado**: 1:N (um evento gera vários resultados)

Implementação no Enterprise Architect

Configurações Recomendadas:

1. **Package Structure**:
    - `br.pacehub.domain` (classes de domínio)
    - `br.pacehub.enums` (enumerações)
2. **Stereotypes**:
    - `<<abstract>>` para Usuario
    - `<<enumeration>>` para enums
    - `<<domain>>` para classes de domínio
3. **Visibility**:
    - Atributos: `protected` (#)
    - Métodos: `public` (+)
    - Métodos auxiliares: `private` (-)

Navegability:

- Todas as associações são bidirecionais por padrão
- Composições indicadas com losango preenchido
- Agregações com losango vazio quando aplicável