Relatório do Sistema de Gestão de Espaços Físicos

Integrantes:

Daniel de Oliveira Lira, 241025505

Mateus Rodrigues Barreto, 241011466

Aplicação de Pilares da Orientação a Objetos

Associações: Associações foram criadas com a finalidade de representar ligações entre objetos do mundo real ou conectar módulos entre si, e essas associações foram implementadas principalmente na forma de atributos de classes que são outras classes.

Segue abaixo alguns exemplos:

1. Equipamentos em EspaçoFisico

```java
//Atributos da Classe EspacoFisico
private int capacidade;
private String localizacao,
         tipo; //Sala de Aula, Laboratório ou Sala de Estudos
private LocalTime horarioInicialDisponivel,
            horarioFinalDisponivel;
private Equipamento[] equipamentos;
private transient List<Agendamento> agendamentos = new ArrayList<>();
```

Nesse exemplo, o atributo equipamentos da classe EspacoFisico é na verdade um vetor de elementos do tipo Equipamento, os quais representam o conjunto de equipamentos do espaço em questão.

2. Agendamentos em Usuario e EspacoFisico

```java
//Atributos da Classe Usuario
private String nome,
               email,
               senha,
               telefone;
private transient List<Agendamento> agendamentos = new ArrayList<>();
```

```java
//Atributos da Classe EspacoFisico
private int capacidade;
private String localizacao,
               tipo; //Sala de Aula, Laboratório ou Sala de Estudos
private LocalTime horarioInicialDisponivel,
                  horarioFinalDisponivel;
private Equipamento[] equipamentos;
private transient List<Agendamento> agendamentos = new ArrayList<>();
```

```java
public record Agendamento(LocalDateTime dataInicio, LocalDateTime dataFim, Usuario usuario, EspacoFisico espaco) {
  public boolean sobrepoe(LocalDateTime inicio, LocalDateTime fim) {
    return dataInicio.isBefore(fim) && dataFim.isAfter(inicio);
  }
}
```

O atributo agendamentos é presente tanto em Usuario quanto em EspaçoFisico e representa uma lista de todos os agendamentos da classe em questão.

3. AgendamentoParcial e Agendamento

```java
public record AgendamentoParcial(Agendamento agendamento, LocalTime inicio, LocalTime fim) {}
```

AgendamentoParcial possui como parâmetros um agendamento e outros dois tipos da classe LocalTime, associando essas classes na sua estrutura. AgendamentoParcial representa a parte do agendamento referente a um período específico.

Herança: A herança foi principalmente aplicada na definição das classes bases dentro do pacote entidade, a classe abstrata Usuário é superclasse de todas as outras classes que representam pessoas no domínio da aplicação, como a classe Aluno. Além disso, a classe abstrata Servidor é subclasse de Usuário mas também é superclasse das classes Professor e Administrativo, generalizando o atributo matriculaInstitucional e permitindo uma melhor legibilidade do código. Herança também foi aplicada na definição das exceções, tendo em vista que todas herdam de RuntimeException.

Polimorfismo por Sobrescrita: O projeto lança mão de **polimorfismo por sobrescrita** principalmente na definição do método abstrato getIdentificação() na classe abstrata Usuario e sua implementação especializada nas subclasses Aluno e Servidor, com resultados diferentes de acordo com a instância do objeto na qual o método é chamado.

```java
public abstract String getIdentificacao();
```

```java
    // Aluno
    public String getIdentificacao() {
      return getMatricula();
    }
```

```java
    // Servidor
    public String getIdentificacao() {
      return getMatriculaInstitucional();
    }
```

getIdentificação() em aluno retorna sua matrícula, e getIdentificação() em professor retorna sua matriculaInstitucional.

Além disso, **polimorfismo por sobrescrita** também é aplicado na sobrescrita dos métodos equals() e hashCode(), os quais são sobrescritos da sua implementação comum na superclasse Objeto para permitir uma melhor compatibilidade com os ArrayLists utilizados na classe Registro e outras classes do projeto.

```java
  //Sobrescrita dos Métodos equals e hashCode
  @Override
  public boolean equals(Object objeto) {
    if (this == objeto) {
      return true;
    }
    if (objeto == null || getClass() != objeto.getClass()) {
      return false;
    }
    Servidor servidor = (Servidor) objeto;
    return matriculaInstitucional.equals(servidor.matriculaInstitucional);
  }

  @Override
  public int hashCode() {
    return matriculaInstitucional.hashCode();
  }
```

Polimorfismo por Sobrecarga: O **polimorfismo por sobrecarga** ocorre primariamente na definição dos métodos construtores das classes em geral, e também na exceção ForaDoIntervaloException(), a qual em função de sua assinatura, lança mensagens diferentes.

```java
public class ForaDoIntervaloException extends RuntimeException{
  public ForaDoIntervaloException(int min, int max) {
    super("O intervalo deve estar entre " + min + " e " + max);
  }
  public ForaDoIntervaloException(String mensagem) {
    super(mensagem);
  }
}
```

Quando são informados inteiros limitantes, a exceção formata automaticamente uma mensagem explicitando os limites a serem inseridos corretamente, e, no outro caso, a exceção tem um caráter mais genérico e deve ser informada uma String que contenha a mensagem contextualizada a ser enviada.

Polimorfismo por Coerção: O **polimorfismo por coerção** foi inicialmente implementado em algumas ocasiões pontuais como para escolher entre obter a matricula do aluno ou a matriculaInstitucional de um Servidor com base na verificação do tipo, conforme segue exemplo abaixo:

```java
private static String obterIdentificacao(Usuario usuario) {
    if (usuario instanceof Aluno aluno) {
        return aluno.getMatricula();
    } else if (usuario instanceof Servidor servidor) {
        return servidor.getMatriculaInstitucional();
    }
    return usuario.getNome();
}
```

No entanto, posteriormente foram observadas possíveis refatorações mais simples, como o próprio método abstrato getIndentificação(), e o polimorfismo por sobrescrita foi aposentado.

Polimorfismo Paramétrico: O **polimorfismo paramétrico** foi extensivamente utilizado na forma de Lists e ArrayLists, pois estas facilitam em muito o trabalho com conjuntos de dados, garantindo comodidade e conveniências que não seriam possíveis caso fossem apenas utilizados os clássicos arrays do Java. A classe Registro, responsável por salvar em uma memória temporária do sistema as entidades cadastradas, é um exemplo desse tipo de Polimorfismo, em que seus atributos são implementados na forma de ArrayLists.

```java
//Atributos da Classe
private static ArrayList<Aluno> alunos = new ArrayList<>();
private static ArrayList<Servidor> servidores = new ArrayList<>();
private static ArrayList<EspacoFisico> salasDeAula = new ArrayList<>();
private static ArrayList<EspacoFisico> laboratorios = new ArrayList<>();
private static ArrayList<EspacoFisico> salasDeEstudos = new ArrayList<>();
```

Exceções

Exceções foram essenciais para garantir o bom andamento do projeto auxiliando em funções como validação e verificação de entradas. Abaixo está um exemplo de lançamento de uma exceção personalizada:

```java
private static void verificarExistenciaEspacoFisico(EspacoFisico espaco) {
  if (espaco == null) {
    throw new EspacoFisicoNaoExiste();
  }
}
```

Cada exceção criada teve um propósito específico para condições que não cabiam em exceções tradicionais, ou para deixar o projeto mais limpo. Segue abaixo, lista com todas as exceções personalizadas implementadas e sua explicação:

- **CampoVazioException:** Exceção genérica personalizada lançada quando há campos em branco.
- **DataFuturaException:** Exceção genérica personalizada lançada quando a data não está no futuro.
- **DatallegalException:** Exceção personalizada lançada quando o usuário tenta agendar em uma data diferente da de um agendamento futuro já existente.
- **DiasExcedidosException:** Exceção personalizada lançada quando o usuário tenta agendar por mais dias do que o permitido.
- **EmailAlunoFormatoInvalidoException:** Exceção personalizada lançada quando o e-mail de aluno não segue o padrão aceito (matricula@aluno.unb.br ou matricula@estudante.unb.br).
- **EmailDuplicadoException:** Exceção personalizada lançada quando o e-mail já está registrado no sistema.
- **EmailServidorFormatoInvalidoException:** Exceção personalizada lançada quando o e-mail do servidor não segue o formato nome.sobrenome@unb.br.
- **EquipamentoDuplicadoException:** Exceção personalizada lançada quando um equipamento já está cadastrado na sala.
- **EspacoFisicoNaoExiste:** Exceção personalizada lançada quando o espaço físico especificado não existe.
- **ForaDoIntervaloException:** Exceção personalizada lançada quando um valor fornecido está fora do intervalo permitido.
- **HorarioIndisponivelException:** Exceção personalizada lançada quando o horário selecionado está sendo ocupado por outro usuário.
- **HorarioNaoElegivelException:** Exceção personalizada lançada quando o horário selecionado não está dentro da faixa permitida para agendamentos.
- **LocalizacaoDuplicadaException:** Exceção personalizada lançada quando uma localização já está cadastrada no sistema.
- **LoginInvalidoException:** Exceção personalizada lançada quando o e-mail ou senha informados são inválidos.
- **MatriculaDuplicadaException:** Exceção personalizada lançada quando a matrícula informada já está registrada no sistema.

- **MatriculaFormatoInvalidoException:** Exceção personalizada lançada quando a matrícula não possui exatamente 9 dígitos.
- **PeriodoInvalidoException:** Exceção personalizada lançada quando a data final do agendamento é anterior ou igual à data inicial.
- **PeriodoMinimoException:** Exceção personalizada lançada quando o agendamento tem duração inferior ao mínimo exigido.
- **SenhaFormatoInvalidoException:** Exceção personalizada lançada quando a senha não segue o formato esperado.
- **TelefoneDuplicadoException:** Exceção personalizada lançada quando o número de telefone já está cadastrado no sistema.
- **TipoInesperadoException:** Exceção personalizada lançada quando o tipo de dado informado não é o esperado.
- **TipoInteiroEsperadoException:** Exceção personalizada lançada quando um valor não inteiro é fornecido em um campo que exige inteiros.