Commit 5039bdb - 23/11/23

ALUNO.JAVA

A classe `Aluno` parece está bem estruturada e alinhada com o enunciado fornecido. No entanto, aqui estão alguns insights e sugestões para melhorar ainda mais a implementação:

1. **Validação de Dados:**
   - Considere adicionar validações nos métodos `addTurma`, `addDisciplina`, `addSala` para verificar se os objetos passados como parâmetros não são nulos.
2. **Método `realocarTurma`:**
   - Atualmente, o método `realocarTurma` remove a turma atual e adiciona a nova turma. No entanto, pode ser útil adicionar uma verificação adicional para garantir que a nova turma não exceda o número máximo de alunos.
3. **Método `removerTurma`:**
   - Assim como no método `realocarTurma`, você pode adicionar uma verificação para garantir que a turma a ser removida não ultrapasse o número mínimo de alunos. Além disso, o método poderia retornar um booleano indicando se a remoção foi bem-sucedida ou não.
4. **Método `getTurmas`:**
   - O método `getTurmas` parece funcionar bem, mas talvez seja interessante adicionar mais informações sobre cada turma, como os alunos matriculados em cada uma delas.
5. **Construtor:**
   - Considere remover os parâmetros `ArrayList<Turma> turma`, `ArrayList<Disciplina> disciplina`, `ArrayList<Sala> salas` do construtor, pois eles podem ser inicializados vazios e adicionados posteriormente com os métodos adequados.
6. **Mensagens de Saída:**

- As mensagens de saída do console são úteis para depuração, mas considere substituí-las por exceções apropriadas para que o usuário da classe possa tratar erros de maneira mais adequada.

Além desses pontos, é importante garantir que a interação entre as classes (`Aluno`, `Turma`, `Disciplina`, etc.) esteja de acordo com o que é esperado pelo enunciado. Certificar-se de testar a classe em diferentes cenários para garantir seu correto funcionamento.

APP.JAVA

A classe `App` atualmente tem apenas um `println` indicando que haverá uma interface gráfica. Para atender aos requisitos do trabalho, será preciso criar uma interface gráfica usando o Swing. Aqui estão algumas sugestões para começarmos:

```
import javax.swing.*;

public class App {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Sistema de Gerenciamento Escolar");
            frame.setSize(800, 600);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            // Adicione seus componentes gráficos (botões, campos de texto, etc.) aqui

            frame.setVisible(true);
        });
    }
}
```

Aqui estão algumas sugestões para expandir a interface gráfica:

1. **Painéis e Layouts:**
    - Use painéis (JPanel) para organizar os componentes.
    - Escolha um layout apropriado (FlowLayout, BorderLayout, GridLayout, etc.) para organizar os componentes dentro dos painéis.
2. **Botões e Ações:**
    - Adicione botões para representar ações (por exemplo, cadastrar aluno, cadastrar disciplina, matricular aluno, etc.).
    - Associe ações aos botões usando `ActionListener`.

3. **Campos de Texto:**
    - Adicione campos de texto para entrada de dados (por exemplo, nome do aluno, código da disciplina, etc.).
4. **Listas ou Tabelas:**
    - Considere o uso de listas ou tabelas para exibir informações como turmas de um aluno, disciplinas cadastradas, etc.
5. **Mensagens de Feedback:**
    - Adicione áreas para exibir mensagens de feedback ao usuário (sucesso, erro, etc.).
6. **Tratamento de Eventos:**
    - Implemente o tratamento de eventos para responder às ações do usuário (por exemplo, clique em um botão).

Essa estrutura básica pode ser expandida e personalizada conforme as necessidades específicas do seu sistema. Lembre-se de dividir a lógica de apresentação da lógica de negócios, seguindo os princípios da programação orientada a objetos.

CURSO.JAVA

A classe `Curso` parece bem estruturada. Aqui estão algumas observações e sugestões:

1. **Método `adicionaAluno`:**
    - Atualmente, o método `adicionaAluno` adiciona um aluno diretamente à lista de alunos. Considerando adicionar verificações adicionais, como verificar se o curso atingiu o limite de alunos antes de adicionar um novo aluno.
2. **Campo `LIMITE_ALUNOS`:**
    - O campo `LIMITE_ALUNOS` é uma constante que define o limite de alunos para o curso. Certificar-se de que este valor está de acordo com os requisitos do sistema. Talvez seja interessante torná-lo ajustável (por exemplo, por meio de um método `setLimiteAlunos`) caso o limite possa ser alterado dinamicamente.
3. **Método `getVagasRestantes`:**

- O método `getVagasRestantes` calcula as vagas restantes subtraindo o número atual de alunos do limite total. Certifico que este cálculo está correto e em conformidade com os requisitos do sistema.

4. **Inicialização da Lista de Alunos:**
   - No construtor, é inicializando a lista de alunos como `null`, mas logo em seguida a inicializa como um novo `ArrayList`. Pode ser mais consistente inicializá-la diretamente como um `ArrayList` vazio.

Aqui está uma versão revisada considerando essas sugestões:

```
import java.util.ArrayList;

public class Curso {
    private String titulo;
    private final int LIMITE_ALUNOS = 200;
    private ArrayList<Aluno> alunos = new ArrayList<>();

    public Curso(String titulo) {
        this.titulo = titulo;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAlunos() {
        return alunos.toString();
    }

    public void setAlunos(ArrayList<Aluno> alunos) {
        this.alunos = alunos;
    }

    public void adicionaAluno(Aluno aluno) {
        if (aluno != null && alunos.size() < LIMITE_ALUNOS) {
            alunos.add(aluno);
        } else {
            System.out.println("Limite de alunos atingido. Não foi possível adicionar
o aluno.");
        }
    }

    public int getAlunosVinculados() {
        return alunos.size();
    }
```

```
    public int getVagasRestantes() {
        return LIMITE_ALUNOS - alunos.size();
    }
}
```

Essas alterações tornam o código mais claro e robusto, levando em consideração verificações adicionais e melhor inicialização da lista de alunos.

DISCIPLINA.JAVA

A classe `Disciplina` parece bem estruturada. Aqui estão algumas observações e sugestões:

1. **Atributo `turmas`:**
    - A classe declara um array de `Turma[] turmas`, mas esse array não está sendo utilizado no construtor e nos métodos getters e setters. Se a intenção era ter um array de turmas associadas à disciplina, seria necessário ajustar o código para utilizar esse atributo.
2. **Validação de Dados:**
    - Considere adicionar verificações nos métodos setters para garantir que dados essenciais, como título, código e carga horária, não sejam nulos ou negativos, dependendo dos requisitos do sistema.
3. **Inicialização de Arrays:**
    - Certifique-se de que os arrays de professores, salas e turmas sejam inicializados adequadamente no construtor, evitando problemas de referência nula.

Aqui está uma versão revisada considerando essas sugestões:

```
import java.util.Arrays;

public class Disciplina {
    private String titulo;
    private Integer codigo;
    private Integer cargaHoraria;
    private String descricao;
    private Integer aulasSemana;

    private Professor[] professoresMinistrantes;
    private Sala[] salas;
    private Turma[] turmas;
```

```
    // Construtor da classe Disciplina
    public Disciplina(String titulo, Integer codigo, Integer cargaHoraria, String desc
ricao, Integer aulasSemana,
            Professor[] professoresMinistrantes, Sala[] salas, Turma[] turmas) {
        this.titulo = (titulo != null) ? titulo.toUpperCase() : null;
        this.codigo = (codigo != null && codigo > 0) ? codigo : null;
        this.cargaHoraria = (cargaHoraria != null && cargaHoraria > 0) ? cargaHoraria
: null;
        this.descricao = (descricao != null) ? descricao.toUpperCase() : "Não possui d
escrição.";
        this.aulasSemana = (aulasSemana != null && aulasSemana > 0) ? aulasSemana : nu
ll;

        // Inicialização dos arrays
        this.professoresMinistrantes = (professoresMinistrantes != null) ? Arrays.copy
Of(professoresMinistrantes, professoresMinistrantes.length) : null;
        this.salas = (salas != null) ? Arrays.copyOf(salas, salas.length) : null;
        this.turmas = (turmas != null) ? Arrays.copyOf(turmas, turmas.length) : null;
    }

    // Métodos getters e setters

    @Override
    public String toString() {
        return "Disciplina [titulo=" + titulo + ", codigo=" + codigo + ", cargaHoraria
=" + cargaHoraria + ", descricao="
                + descricao + ", aulasSemana=" + aulasSemana + ", professoresMinistran
tes="
                + Arrays.toString(professoresMinistrantes) + ", salas=" + Arrays.toStr
ing(salas) + ", turmas="
                + Arrays.toString(turmas) + "]";
    }
}
```

Essas alterações visam garantir a consistência e validação adequada dos dados da disciplina.

MAIN.JAVA

1. **Variáveis Estáticas:**
    - Usamos muitas variáveis estáticas, como `idMapProf` e `idMapAluno`. Isso pode causar problemas em ambientes multithread. Vamos reconsiderar se elas realmente precisam ser estáticas ou se podem ser parte do estado de uma instância da classe `Main`.
2. **Cadastro de Alunos:**
    - Começamos a implementação do método `cadastroAluno`, mas parece que a implementação real está faltando. Vamos revisitar isso para garantir que

esteja completo.

3. **Validação de Entrada:**
   - Notamos que não validamos completamente a entrada do usuário. Precisamos adicionar mais verificações e feedback para garantir um comportamento mais robusto, considerando que a entrada do usuário pode ser imprevisível.
4. **Manipulação de Mapas:**
   - Manipulamos diretamente os mapas (`alunos`, `turmas`, etc.) no código principal. Vamos considerar encapsular a lógica de manipulação desses mapas em métodos separados para melhorar a legibilidade e manutenção do código.
5. **Organização do Código:**
   - Achamos que nosso código pode se beneficiar de métodos menores e especializados. Vamos organizar melhor, especialmente no método `main`, para tornar o código mais fácil de entender e manter.
6. **Comentários no Método `realocacaoAluno`:**
   - Percebemos que os comentários mencionam "turma atual" e "nova turma", mas o método `realocarTurma` na classe `Aluno` é usado para realizar a realocação. Vamos garantir que nossos comentários correspondam à lógica real do código.
7. **Lógica de Matrícula do Aluno em um Curso:**
   - O método `matriculaAluno` faz referência a uma lista de cursos (`ArrayList<Curso> cursos`), mas parece que esta lista está vazia ou não inicializada no código fornecido. Precisamos garantir que a lista de cursos seja populada e usada corretamente.
8. **Concorrência em Ambientes Multithread:**
   - Considerando que nosso código pode ser usado em um ambiente multithread, vamos adotar estratégias para garantir a segurança da thread, como o uso de construções de sincronização.
9. **Implementação Incompleta:**
   - Alguns métodos, como `cadastroProfessor`, `alocacaoProfessor`, `realocacaoProfessor`, `visualizarProfessores`, `listarProfessores`, `dadosProfessor`, `editarProfessor`, `visualizarAluno`, `listarAlunos`, `dadosAluno`, e `editarAluno`,

têm suas assinaturas declaradas, mas as implementações estão faltando. Vamos garantir que esses métodos sejam implementados conforme necessário.

PROFESSOR.JAVA

1. **Construtor:**
   - Parece que há um pequeno erro no construtor. A verificação `if (emailAcad == null && super.getNome() != null)` está utilizando `emailAcad`, que não é passado como parâmetro, em vez de `email`. Vamos corrigir isso para garantir que o e-mail acadêmico seja gerado corretamente.
2. **Lógica de Atribuição do E-mail Acadêmico:**
   - A lógica para atribuição do e-mail acadêmico pode ser simplificada e melhorada. Podemos fazer isso diretamente no construtor, evitando a necessidade do bloco condicional `if`.
3. **Verificação de Disciplinas Alocadas:**
   - A verificação `if (disciplinas != null)` está correta para verificar se o array de disciplinas foi passado. No entanto, logo em seguida, definimos `this.alocado` como `true` independentemente da condição. Vamos corrigir isso para refletir a lógica correta.
4. **Atribuição do E-mail Acadêmico:**
   - A atribuição do e-mail acadêmico atualmente usa `emailAcad`, que não é passado como parâmetro. Vamos corrigir isso para usar o parâmetro `email`.

Aqui está o código ajustado:

```
import java.util.Arrays;

public class Professor extends Usuario {
    private String emailAcad;
    private String escola;
    private Boolean alocado = false;
    private Disciplina[] disciplinas;

    public Professor(String nome, Long cpf, Integer matricula, String email, Disciplin
a[] disciplinas,
                     String escola, Boolean alocado) {

        super(nome, cpf, matricula, email);
```

```
        // Corrigindo a lógica de atribuição do e-mail acadêmico
        this.emailAcad = (email != null) ? email : (super.getNome() + "@educacorp.co
m");

        this.escola = escola;

        // Corrigindo a lógica de atribuição de 'alocado'
        this.alocado = (disciplinas != null) && (disciplinas.length > 0);

        this.disciplinas = disciplinas;
    }

    // Restante do código permanece inalterado

    @Override
    public String toString() {
        return super.toString() + "\\nTipo: Professor" +
                "\\nEmailAcad: " + emailAcad + "\\nEscola: " + escola +
                "\\nAlocado: " + alocado + "\\nDisciplinas: " + Arrays.toString(discip
linas);
    }
}
```

Essas alterações garantem que a inicialização dos atributos seja feita corretamente, considerando as condições e lógicas adequadas.

SALA.JAVA

Vamos fazer algumas melhorias no código:

1. **Atributo `vagas`:**
    - Atributo `vagas` parece ser redundante, pois está sendo inicializado com `super.getVagas()`. Vamos usar o parâmetro `vagas` diretamente no construtor.
2. **Atributo `profMinistrante`:**
    - Se estamos passando um array de professores ministrantes no construtor da superclasse, podemos assumir que o primeiro professor desse array será o professor ministrante. Vamos ajustar isso.
3. **Remoção de Atributo Redundante:**
    - Parece que o atributo `possuiProfessor` é redundante, pois podemos verificar se o `profMinistrante` é nulo para determinar se a sala possui um professor. Vamos remover esse atributo.

Aqui está o código ajustado:

```
import java.util.Arrays;

public class Sala extends Turma {
    private Integer numSala;
    private Professor profMinistrante;
    private String[] horarios;

    public Sala(String titulo, Integer codigo, Integer cargaHoraria, String descricao,
Integer aulasSemana,
            Professor[] professoresMinistrantes, Sala[] salas, Sala[] labs, Integer va
gas, Integer numSala,
            String[] horarios) {
        super(titulo, codigo, cargaHoraria, descricao, aulasSemana, professoresMinistr
antes, salas, labs);
        this.numSala = numSala;

        // Definindo o primeiro professor do array como o professor ministrante
        this.profMinistrante = (professoresMinistrantes != null && professoresMinistra
ntes.length > 0) ?
                professoresMinistrantes[0] : null;

        this.horarios = horarios;
    }
}
```

Essas alterações tornam o código mais limpo e eliminam redundâncias desnecessárias.

TURMA.JAVA

Alguns pontos que podem ser melhorados ou corrigidos:

1. No construtor da classe `Turma`, a variável `vagas` está sendo atribuída antes de ser inicializada. Além disso, a condição para verificar se a turma está lotada parece estar incorreta. Aqui está uma versão ajustada:

```
public Turma(String titulo, Integer codigo, Integer cargaHoraria, String descricao, In
teger aulasSemana,
            Professor[] professoresMinistrantes, Sala[] salas, Sala[] labs) {
    super(titulo, codigo, cargaHoraria, descricao, aulasSemana, professoresMinistrante
s, salas, labs);

    this.vagas = MAX_VAGAS;
    this.lotada = false;

    // A condição correta seria verificar se o número de alunos já matriculados é igua
l ao máximo de vagas
    if (this.alunos.size() >= this.vagas) {
        setLotada();
```

```
    }
}
```

1. No método `setVagasRestantes()`, a fórmula parece estar invertida. Aqui está uma correção:

```
public Integer getVagasRestantes() {
    return this.MAX_VAGAS - this.vagas;
}
```

Esses ajustes devem melhorar o funcionamento do código.

USUÁRIO.JAVA

O código parece estar bem estruturado! No entanto, aqui estão algumas sugestões para melhorá-lo:

1. **Constantes para Números Mágicos:**
    - Evitar números mágicos em seu código. Por exemplo, o valor `100` e `900` em `gerarMatricula()` podem não ser autoexplicativos. Você pode considerar a criação de constantes para esses valores.
2. **Método `toString()`:**
    - O método `toString()` é bem implementado, mas você pode adicionar a anotação `@Override` para indicar explicitamente que você está substituindo o método da classe `Object`.

Aqui está o código ajustado considerando essas sugestões:

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public abstract class Usuario {
    static final Set<Integer> matriculasGeradas = new HashSet<>();
    private String nome;
    private Long cpf;
    private Integer matricula;
    private String email;

    public Usuario(String nome, Long cpf, Integer matricula, String email) {
        this.nome = nome;
        this.cpf = cpf;
```

```
        if (matricula == null) {
            this.matricula = gerarMatricula();
        } else {
            this.matricula = matricula;
        }
        this.email = email;
    }

    public int gerarMatricula() {
        Random random = new Random();
        int matricula;

        do {
            matricula = 100 + random.nextInt(900); // Gera um número aleatório de 3 dí
gitos
        } while (matriculasGeradas.contains(matricula));

        matriculasGeradas.add(matricula);
        return matricula;
    }

    public String getNome() {
        return nome;
    }

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

    public Long getCpf() {
        return cpf;
    }

    public void setCpf(Long cpf) {
        this.cpf = cpf;
    }

    public Integer getMatricula() {
        return matricula;
    }

    public void setMatricula(Integer matricula) {
        this.matricula = matricula;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
```

```
        return "Usuario - Visão Geral \\n" +
                "Nome: " + nome + "\\nCPF: " + cpf +
                "\\nMatricula: " + matricula + "\\nEmail: " + email;
    }
}
```

Essas são sugestões mínimas e opcionais. O código já está bem escrito!