Escola de Artes, Ciências e Humanidades - EACH-USP
ACH2002 - Introdução à Ciência da Computação II
EACH - Segundo Semestre de 2010
Primeiro Exercício Prático - Data de entrega: até 26 de setembro 2010

ESCOLA

Neste trabalho você deverá implementar algumas classes para gerenciar uma escola.

O sistema será compostos pelas seguintes classes e interfaces (muitas delas já implementadas, disponibilizadas juntamente com o código do EP). As classes hachuradas são aquelas que você precisará implementar/completar.

Nome	Tipo	Resumo
Estudante	classe abstrata	Classe abstrata que contém algumas constantes (para definir o tipo do estudante) bem como os métodos abstratos que devem ser implementados pelas subclasses.
InterfaceDedicado	interface	Interface que contém a assinatura dos métodos que definem um estudante dedicado.
InterfaceAtleta	interface	Interface que contém a assinatura dos métodos que definem um atleta.
EstudanteRegular	classe	A classe EstudanteRegular é subclasse de Estudante.
EstudanteAtleta	classe	A classe EstudanteAtleta é subclasse de Estudante e implementa a interface InterfaceAtleta.
EstudanteDedicado	classe	A classe EstudanteDedicado é subclasse de Estudante e implementa a interface InterfaceDedicado.
SuperEstudante	classe	A classe SuperEstudante é subclasse de Estudante e implementa as interfaces InterfaceAtleta e InterfaceDedicado.
InterfaceSala	interface	Interface que contém a assinatura dos métodos que definem uma sala.
Sala	classe	Classe que implementa a interface InterfaceSala.
SalaEspecial	classe	A classe SalaEspecial é subclasse da classe Sala.
Escola	classe	Classe que possui os métodos para controlar as salas e os estudantes dentro das salas.
ExecutaEscola	classe	Classe “executável” (que possui método *main*) que pode ser usada para testar as demais classes.

A seguir cada uma das classes e tarefas será detalhada.

Classe Estudante

Possui quatro constantes estáticas do tipo inteiro: *TIPO_ESTUDANTE_REGULAR*, *TIPO_ESTUDANTE_ATLETA*, *TIPO_ESTUDANTE_DEDICADO* e *TIPO_SUPER_ESTUDANTE* que definem valores para o atributo *tipo* de um estudante.

O construtor da classe recebe como parâmetro um inteiro correspondente ao tipo do estudante que está sendo instanciado, esse valor deve ser guardado no atributo *tipo*.

Possui dois métodos abstratos e dois métodos não abstratos:

abstract String nomeDoTipoDeEstudante() método a ser implementado nas subclasses e que deve retornar uma *String* correspondente ao tipo do estudante: **"regular"** para EstudanteRegular, **"atleta"** para EstudanteAtleta, **"dedicado"** para EstudanteDedicado e **"super"** para SuperEstudante (NOTE: retorne apenas esses valores: apenas o nome do tipo do estudante sem espaços em branco, todas as letras em minúscula e sem nenhum outro caractere);

abstract boolean implicaCom(Estudante e) método a ser implementado nas subclasses que deve retornar *true* caso o estudante atual (o que está implementando o método) implique com o estudante ***e*** (passado como parâmetro). Para a classe EstudanteRegular, a implementação deste método deve retornar sempre *false*. Para a classe EstudanteAtleta este método deverá retornar *true* caso a variável ***e*** contenha um EstudanteDedicado. Para a classe EstudanteDedicado este método deverá retornar sempre *false*. Para o SuperEstudante este método deverá retornar *false* caso a variável ***e*** contenha um SuperEstudante; caso contrário deverá retornar *true*.

int numeroDeHorasQueEstuda() método que retorna a quantidade de horas que o Estudante estuda por semana. Na implementação da classe estudante retorna 10 horas. Este método deve ser redefinido apenas nas classes: EstudanteAtleta para retornar o valor 8; e na classe EstudanteDedicado para retornar 16.

final int retornaTipoDoEstudante() método que retorno o valor do atributo *tipo*.

Interface InterfaceDedicado

Esta interface possui a assinatura de dois métodos **estudarMais()** e **fazerAtividadesExtras()**. A implementação destes métodos deverá, simplesmente, retornar *true*.

Interface InterfaceAtleta

Esta interface possui a assinatura de dois métodos **treinar()** e **fazerExerciciosFisicos()**. A implementação destes métodos deverá, simplesmente, retornar *true*.

Classe EstudanteRegular

Esta classe é subclasse da classe Estudante e não implementa nenhuma interface.

Além de estender a classe Estudante, deve possuir um construtor sem parâmetros **EstudanteRegular()**

Classe EstudanteAtleta

Esta classe é subclasse da classe Estudante e implementa a interface InterfaceAtleta.

Além de estender e implementar os métodos da classe Estudante e da interface InterfaceAtleta, deve possuir um construtor sem parâmetros **EstudanteAtleta()**

Classe EstudanteDedicado

Esta classe é subclasse da classe Estudante e implementa a interface InterfaceDedicado.

Deve possuir um construtor sem parâmetros **EstudanteDedicado()**

Classe SuperEstudante

Esta classe é subclasse da classe Estudante e implementa as interfaces InterfaceDedicado e InterfaceAtleta.

Deve possuir um construtor sem parâmetros **SuperEstudante()**

Interface InterfaceSala

Esta interface possui a assinatura dos seguintes métodos:

boolean estaLotada() método que deve retornar *true* caso a sala esteja lotada (caso o número de estudantes na sala seja igual ao número máximo de estudantes).

boolean adicionarEstudante(Estudante e) método que deve retornar *false* caso a sala esteja lotada; caso contrário, deverá verificar as regras de cada tipo de sala, se for possível adicionar o estudante ***e,*** deverá adicioná-lo na sala (já que uma sala irá possuir um arranjo (*array*) de elementos do tipo Estudante, deverá colocar o estudante passado como parâmetro na primeira posição livre do arranjo), atualizar o contador do número atual de estudantes e retornar *true*. As regras de cada tipo de sala estão descritas nas classes Sala e SalaEspecial.

void imprimirListaDeEstudantes() método que deverá imprimir a lista de estudantes de uma sala. Este método deverá imprimir um estudante por linha e apenas o "nome do tipo do estudante" (valor passado pelo método **nomeDoTipoDeEstudante** presente na classe Estudante.

void aumentarTamanhoDaSala(int aumento) este método deverá aumentar o tamanho máximo da sala no número de posições passadas pela variável *aumento* (esta variável sempre receberá valores maiores que zero), ou seja, criar um arranjo de tamanho maior (igual ao tamanho original + aumento) para guardar os estudantes e atualizar as variáveis relevantes). Note que os estudantes existentes na sala antes do aumento devem estar presentes na sala após o aumento.

boolean ehPossivelColocarNestaSala(Estudante[] estudantes) este método deve retornar *true* caso seja possível adicionar na sala atual todos os estudantes passados como parâmetro no arranjo *estudantes*. As regras de quais estudantes podem ser adicionados encontram-se nas classes Sala e SalaEspecial. Este método não insere nenhum estudante, apenas verifica se seria possível adicionar esses estudantes a sala atual.

Classe Sala

Esta classe implementa a interface InterfaceSala. Ela possui os seguintes atributos:

```
int numeroMaximoDeEstudantes; // número máximo de estudantes que cabem na sala
int numeroAtualDeEstudantes; // número de estudantes atualmente na sala
Estudante[] estudantes; // arranjo contendo os estudantes que estão nesta sala
```

O **construtor** da classe Sala deverá receber como parâmetro o número máximo de estudantes que cabem na sala. Este construtor deverá inicializar todos os atributos desta classe.

A **regra** de adição de estudantes da classe Sala é a seguinte: caso a sala não esteja lotada poderão ser adicionados estudantes apenas do mesmo tipo (cada sala [um objeto do tipo Sala] poderá receber apenas estudantes de um mesmo tipo). Quando a sala estiver vazia, ela poderá aceitar estudantes de qualquer tipo (mas os demais estudantes deverão ser do mesmo tipo do primeiro). Esta regra deverá ser usada na implementação dos métodos **boolean adicionarEstudante(Estudante a)** e **boolean ehPossivelColocarNestaSala(Estudante[] estudantes)**

Classe SalaEspecial

Esta classe é subclasse da classe Sala.

O **construtor** da classe SalaEspecial deverá receber como parâmetro o número máximo de estudantes que cabem na sala e chama o construtor de sua super classe.

A **regra** de adição de estudantes da classe SalaEspecial é a seguinte: caso a sala não esteja lotada poderão ser adicionados estudantes apenas que não impliquem uns com os outros. Por exemplo, uma SalaEspecial pode conter EstudantesRegulares e EstudantesDedicados, mas não EstudantesAtletas e EstudantesDedicados. Para verificar se um estudante implica ou não com outro utilize o método **boolean implicaCom(Estudante e)** da classe Estudante. Esta regra deverá ser usada na implementação dos métodos **boolean adicionarEstudante(Estudante a)** e **boolean ehPossivelColocarNestaSala(Estudante[] estudantes)**.

Classe Escola

Esta classe possui o seguinte atributo:

```
private InterfaceSala[] salas; // contém um arranjo das salas da escola
```

O construtor da classe Escola deverá receber como parâmetro um inteiro indicando quantas Salas existem nesta escola e criar um arranjo com esse número de salas no atributo *salas*.

Esta classe possui os seguintes métodos:

`public boolean adicionarSala(int posicaoNoArranjo, int numeroMaximoDeEstudantes, boolean ehSalaEspecial)` este método recebe três parametros: *posicaoNoArranjo* que indica em que posição do arranjo *salas* uma nova sala deverá ser criada; *numeroMaximoDeEstudantes* que indica o número máximo de estudantes que caberão na nova sala; e *ehSalaEspecial* que indica se a nova sala deverá ser do tipo Sala ou do tipo SalaEspecial. Este método funciona da seguinte maneira: ele deverá retornar *false* caso a variável *posicaoNoArranjo* corresponda a um índice inválido (menor que zero ou maior ou igual à quantidade de salas) **OU** também deverá retornar *false* caso já exista uma Sala na posição *posicaoNoArranjo* do arranjo *salas* (ou seja, não deve sobrepor uma sala já existente). Caso contrário deverá inserir uma nova sala na posição indicada (do tipo e tamanho indicados pelos parâmetros) e retornar *true*.

`public boolean removerSala(int posicaoNoArranjo)` este método deve remover a sala da posição *posicaoNoArranjo* (ou seja, colocar **null** nessa posição do arranjo) e retornar *true* **caso** essa posição seja válida **E** exista uma sala nessa posição. **Caso contrário** deverá retornar *false*.

`public boolean inserirEstudanteNaSala(int posicaoNoArranjo, Estudante e)` este método deve inserir o Estudante *e* na Sala situada na posição *posicaoNoArranjo* e retornar *true* caso esta adição seja possível. Note que este método deve retornar *false* **caso**: a posição *posicaoNoArranjo* seja inválida, **OU** caso não exista sala nessa posição (salas[posicaoNoArranjo] == null) **OU** caso não seja possível adicionar o Estudante *e* nesta sala (utilize o método `adicionarEstudante(Estudante a)` da classe Sala).

`public void listarEstudantesDaSala(int posicaoNoArranjo)` este método deve chamar o método *imprimirListaDeEstudantes()* da sala situada na *posicaoNoArranjo* caso este índice seja válido e exista um sala nessa posição; caso contrário não deverá fazer nada.

`public boolean verificaSeEhPossivelAdicionarConjuntoDeEstudantesASala(int posicaoNoArranjo, Estudante[] estudantes)` este método deve consultar o resultado do método *ehPossivelColocarNestaSala(Estudante[] estudantes)* da sala situada na *posicaoNoArranjo* caso este índice seja válido e exista um sala nessa posição (e retornar o resultado desse método). Caso contrário, deverá retornar *false*.

`public boolean aumentarTamanhoDaSala(int posicaoNoArranjo, int valorDoAumento)` este método deverá chamar o método aumentarTamanhoDaSala(int valorDoAumento) da sala situada na posição *posicaoNoArranjo* e retornar *true* caso este índice seja válido e caso haja uma sala nesta posição. Caso contrário, deverá retornar *false*.

Classe ExecutaEscola

Esta classe possui o método main e pode ser usada para testar partes do EP.

Note que esta classe não testa exaustivamente todos os métodos implementados (um teste mais cuidadoso é de responsabilidade do aluno).

Regras de Elaboração e Submissão

Este trabalho é individual, cada aluno deverá implementar e submeter via Col sua solução.

A submissão será feita via um arquivo zip (o nome do arquivo deverá ser o número USP do aluno, por exemplo 1234567.zip). Este arquivo deverá conter EXATAMENTE os seguintes arquivos: EstudanteRegular.java, EstudanteAtleta.java, EstudanteDedicado.java, SuperEstudante.java, Sala.java, SalaEspecial.java e Escola.java. Observação: no arquivo zip não adicionar (sub-)diretórios ou outros arquivos, apenas esses sete arquivos. Note que estes arquivos são .java (e não .class).

Além deste enunciado, você encontrará na página da disciplina um zip contendo todos os arquivos envolvidos neste trabalho (note que os arquivos a serem implementados também estão disponíveis no site, você precisará apenas completá-los).

Qualquer tentativa de fraude ou cola implicará em nota zero para todos os envolvidos.

Guarde uma cópia do trabalho entregue.

A data de entrega é 26 de setembro (domingo) às 23:50h pelo sistema COL, não serão aceitos trabalhos entregues após esta data (não deixe para submeter na última hora).

Não implemente métodos desnecessários. Não utilize conceitos ou classes que não foram estudados em aula. Só complemente a implementação das classes solicitadas.

Caso o EP não compile a nota do trabalho será zero. É importante que você teste seu trabalho executando a classe ExecutaEscola (note que ela não testa todas as funcionalidades de todas as classes).

Preencha o cabeçalho existente no início do arquivo Escola.java (há espaço para se colocar turma, nome do professor, nome do aluno e número USP do aluno).

Todas as classes pertencem ao pacote ep1.