Tecnologia em Análise e Desenvolvimento de Sistemas

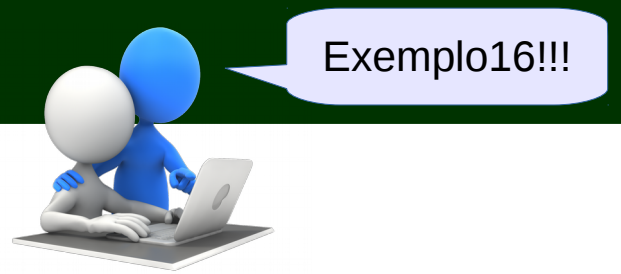
Programação Orientada a Objetos

Coleções

1º/2018

Situação problema

- Um aluno possui quatro notas, prontuário e nome.
- Cada nota equivale a uma porcentagem. As porcentagens devem somar 100%.
- Implementar um sistema que leia as notas de um aluno e apresente a média.




```
package model;
public class Nota {
  private double nota;
  private double porcentagem;
  public Nota(double nota, double porcentagem) {
    this.nota = nota;
    this.porcentagem = porcentagem;
  }
  public double parcial(){
    return nota * (porcentagem/100.0);
  }
  public double getNota() {
    return nota;
  }
  public void setNota(double nota) {
    this.nota = nota;
  }
  public double getPorcentagem() {
    return porcentagem;
  }
  public void setPorcentagem(double porcentagem) {
    this.porcentagem = porcentagem;
  }
}
```

```java
package model;
public class Aluno {
  private int prontuario;
  private String nome;
  private Nota[] notas;
  public Aluno(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new Nota[4];
  }
  public void setNotas(double nota1, double nota2, double nota3,
     double nota4, double peso1, double peso2, double peso3, double peso4){
    if(peso1 + peso2 + peso3 + peso4 == 100){
      notas[0] = new Nota(nota1, peso1);
      notas[1] = new Nota(nota2, peso2);
      notas[2] = new Nota(nota3, peso3);
      notas[3] = new Nota(nota4, peso4);
    }else{
      notas[0] = new Nota(-1, -1);
      notas[1] = new Nota(-1, -1);
      notas[2] = new Nota(-1, -1);
      notas[3] = new Nota(-1, -1);
    }
  }
  public double media(){
    double soma=0;
    for(int i=0; i < 4; i++){
      soma += notas[i].parcial();
    }
    return soma;
  }
```

```java
  public Nota get(int nota){
    Nota retorno = null;
    if(nota >= 0 && nota <= 3){
      retorno = notas[nota];
    }
    return retorno;
  }
  public int getProntuario() {
    return prontuario;
  }
  public void setProntuario(int prontuario) {
    this.prontuario = prontuario;
  }
  public String getNome() {
    return nome;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

```
Nome aluno:
Ednilson
Prontuário:
123
Nota 1:
10
Peso nota 1:
10
Nota 2:
5
Peso nota 2:
40
Nota 3:
10
Peso nota 3:
10
Nota 4:
3
Peso nota 4:
40
Média: 5.2

Process finished with exit code 0
```

```java
package view;
import model.Aluno;
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String nome;
    int prontuario;
    double n1, n2, n3, n4, p1, p2, p3, p4;
    Aluno estudante;
    System.out.println("Nome aluno: ");
    nome = scanner.nextLine();
    System.out.println("Prontuário: ");
    prontuario = scanner.nextInt();
    System.out.println("Nota 1: ");
    n1 = scanner.nextDouble();
    System.out.println("Peso nota 1: ");
    p1 = scanner.nextDouble();
    System.out.println("Nota 2: ");
    n2 = scanner.nextDouble();
    System.out.println("Peso nota 2: ");
    p2 = scanner.nextDouble();
    System.out.println("Nota 3: ");
    n3 = scanner.nextDouble();
    System.out.println("Peso nota 3: ");
    p3 = scanner.nextDouble();
    System.out.println("Nota 4: ");
    n4 = scanner.nextDouble();
    System.out.println("Peso nota 4: ");
    p4 = scanner.nextDouble();
    estudante = new Aluno(prontuario, nome);
    estudante.setNotas(n1, n2, n3, n4, p1, p2, p3, p4);
    System.out.println("Média: " + estudante.media());
  }
}
```

```java
package model;
public class Aluno {
  private int prontuario;
  private String nome;
  private Nota[] notas;
  public Aluno(int prontuario, String nome) {
    this.prontuario = prontuari
    this.nome = nome;
    notas = new Nota[4];
  }
  public void setNotas(double nota1, double nota2, double nota3,
     double nota4, double peso1, double peso2, double peso3, double peso4){
    if(peso1 + peso2 + peso3 + peso4 == 100){
      notas[0] = new Nota(nota1, peso1);
      notas[1] = new Nota(nota2, peso2);
      notas[2] = new Nota(nota3, peso3);
      notas[3] = new Nota(nota4, peso4);
    }else{
      notas[0] = new Nota(-1, -1);
      notas[1] = new Nota(-1, -1);
      notas[2] = new Nota(-1, -1);
      notas[3] = new Nota(-1, -1);
    }
  }
  public double media(){
    double soma=0;
    for(int i=0; i < 4; i++){
      soma += notas[i].parcial();
    }
    return soma;
  }
```

Definiu-se um array

Instanciou-se um array, definiu-se o tamanho da variável.

Acesso aos membros do array é realizado por meio do seu índice (posição que o objeto ocupa no array).

Pode-se acessar os elementos do array por meio do índice, que é um inteiro. Isso permite uma varredura automática da variável array.

Lembrando: Um array é uma variável composto homogênea que utiliza alocação estática de memória, ou seja, uma variável que armazena vários "objetos", organizados sequencialmente na memória.

Está parecendo programação em C. Java pode fazer melhor?

Foreach

- **Para cada** – Pode-se percorrer cada um dos elementos de um conjunto usando o comando "foreach".

```
public double media(){
  double soma=0;
  for(int i=0; i < notas.length; i++){
    soma += notas[i].parcial();
  }
  return soma;
}
```

```
public double media(){
  double soma=0;
  for(Nota nota : notas){
    soma += nota.parcial();
  }
  return soma;
}
```

O resultado final é o mesmo.

Observe que para cada uma das notas no array será feita uma referência nota, e essa pode ser utilizada da forma que for necessário.

varargs

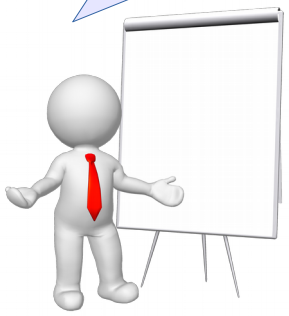
- Lista de Argumentos de Comprimento Variável
- Um tipo de parâmetro seguido por reticências(...) na lista de parâmetros indica que o método recebe um número variável de argumentos desse tipo particular.
- No corpo do método, a lista de parâmetros de tamanho variável é vista como um array (disponibilizando para cada argumento uma posição de armazenamento).

```
public void setNotas(double nota1, double nota2, double nota3, double nota4,
     double peso1, double peso2, double peso3, double peso4){
  if(peso1 + peso2 + peso3 + peso4 == 100){
    notas[0] = new Nota(nota1, peso1);
    notas[1] = new Nota(nota2, peso2);
    notas[2] = new Nota(nota3, peso3);
    notas[3] = new Nota(nota4, peso4);
  }else{
    notas[0] = new Nota(-1, -1);
    notas[1] = new Nota(-1, -1);
    notas[2] = new Nota(-1, -1);
    notas[3] = new Nota(-1, -1);
```

Definiu-se que argumentos é um array do tipo double, mas não sabe-se o tamanho desse array.

O método pode verificar o tamanho do array e tomar sua ação.

Esse exemplo não foi o mais apropriado, veremos outro adiante.



```
public void setNotas(double... argumentos) {
  int soma = 0;
  notas[0] = new Nota(-1, -1);
  notas[1] = new Nota(-1, -1);
  notas[2] = new Nota(-1, -1);
  notas[3] = new Nota(-1, -1);
  if (argumentos.length == 8) {
    for (int i = 4; i < argumentos.length; i++) {
      soma += argumentos[i];
    }
    if (soma == 100) {
      for (int i = 0; i < 4; i++) {
        notas[i].setNota(argumentos[i]);
        notas[i].setPorcentagem(argumentos[i + 4]);
      }
    }
  }
}
```

Outro exemplo (varargs)

- Calcular a média aritmética de vários números.

```
public static final double mediaAritmetica(int... numeros){
  int soma=0;
  for(int i:numeros){
    soma += i;
  }
  return soma / numeros.length;
}
```

A chamada pode ser por meio de um array ou por uma lista de argumentos separados por virgula. O exemplo estão os valores definidos, porém é possível utilizar variáveis do tipo inteiro.

Legal!!!

Na chamada é passado um array?

```
int nros[] = {10, 5, 9, 14, 100, 200};
System.out.println("Media: " + Aluno.mediaAritmetica(nros));
```

```
System.out.println("Media: " + Aluno.mediaAritmetica(10, 5, 9, 14, 100, 200));
```

ArrayList

- É um tipo de objeto que permite armazenar uma **coleção** de dados. Esses dados podem ser acessados por sua posição no ArrayList.
- Muito semelhante a um Array, porém possui algumas vantagens:
  - Não tem tamanho fixo;
  - Gerenciamento da lista (inserção, remoção, etc) é controlado pelo objeto;
  - Suporte ao generics;
  - Outros.

```java
package model;
import java.util.ArrayList;
public class Estudante {
  private int prontuario;
  private String nome;
  private ArrayList<Nota> notas;
  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new ArrayList<>();
  }
  public void addNota(double valor, double porcentagem){
    notas.add(new Nota(valor, porcentagem));
  }
  public double media(){
    double soma=0;
    for(Nota n : notas){
      soma += n.parcial();
    }
    return soma;
  }
  public int getProntuario() {
    return prontuario;
  }
  public void setProntuario(int prontuario) {
    this.prontuario = prontuario;
  }
  public String getNome() {
    return nome;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

```java
package view;
import model.Estudante;
public class Main2 {
  public static void main(String[] args) {
    Estudante estudante;
    estudante = new Estudante(123, "Diego");
    estudante.addNota(5, 15);
    estudante.addNota(10, 5);
    estudante.addNota(3, 80);
    System.out.println(estudante.media());
  }
}
```

```java
package model;
import java.util.ArrayList;
public class Estudante {
  private int prontuario;
  private String nome;
  private ArrayList<Nota> notas;
  public Estudante(int prontuario, S
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new ArrayList<>();
  }
  public void addNota(double valor, double porcentagem){
    notas.add(new Nota(valor, porcentagem));
  }
  public double media(){
    double soma=0;
    for(Nota n : notas){
      soma += n.parcial();
    }
    return soma;
  }
  public int getProntuario() {
    return prontuario;
  }
  public void setProntuario(int p
    this.prontuario = prontuario;
  }
  public String getNome() {
    return nome;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

Definiu-se um ArrayList de Nota

Instância do objeto, notas é uma lista de vários objetos Nota.

```java
package view;
import model.Estudante;
public class Main2 {
  public static void main(String[] args) {
    Estudante estudante;
    estudante = new Estudante(123, "Diego");
    estudante.addNota(5, 15);
    estudante.addNota(10, 5);
    estudante.addNota(3, 80);
    System.out.println(estudante.media());
  }
}
```

Uma das principais vantagens é que não é necessário se preocupar com qual posição armazenar o novo objeto, essa funcionalidade é resolvida pelo ArrayList.

O foreach também resolver o problema com ArrayList. Na verdade o foreach foi criado para ArrayList (Collections) e depois adaptado para array.
Contudo, o método tradicional também funciona!

```java
public double media(){
  double soma=0;
  for(int i=0; i<notas.size(); i++){
    soma += notas.get(i).parcial();
  }
  return soma;
}
```


```java
public void addNota(double valor, double porcentagem){
  notas.add(new Nota(valor, porcentagem));
}
public void addNota(int nota, double valor, double porcentagem){
  notas.add(nota, new Nota(valor, porcentagem));
}
```

```java
public class Main2 {
  public static void main(String[] args) {
    Estudante estudante;
    estudante = new Estudante(123, "Diego");
    estudante.addNota(5, 15);
    estudante.addNota(10, 5);
    estudante.addNota(3, 80);
    System.out.println(estudante.toString());
    System.out.println("Média: " + estudante.media());
  }
}
```

```
Aluno: Diego
Prontuário: 123
Notas:
Valor: 5.0      Porcentagem: 15.0%
Valor: 10.0     Porcentagem: 5.0%
Valor: 3.0      Porcentagem: 80.0%
Média: 3.6500000000000004
```

```java
public class Main2 {
  public static void main(String[] args) {
    Estudante estudante;
    estudante = new Estudante(123, "Diego");
    estudante.addNota(5, 15);
    estudante.addNota(10, 5);
    estudante.addNota(0, 3, 80);
    System.out.println(estudante.toString());
    System.out.println("Média: " + estudante.media());
  }
}
```

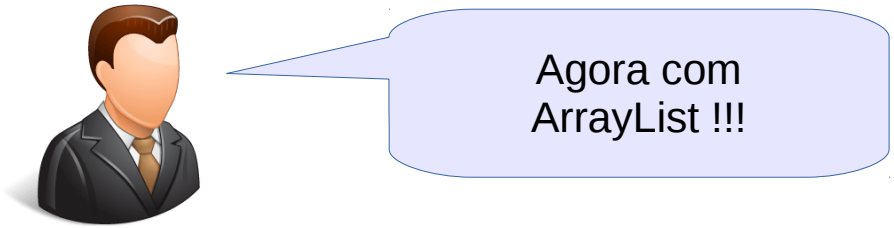
```
Aluno: Diego
Prontuário: 123
Notas:
Valor: 3.0      Porcentagem: 80.0%
Valor: 5.0      Porcentagem: 15.0%
Valor: 10.0     Porcentagem: 5.0%
Média: 3.6500000000000004
```

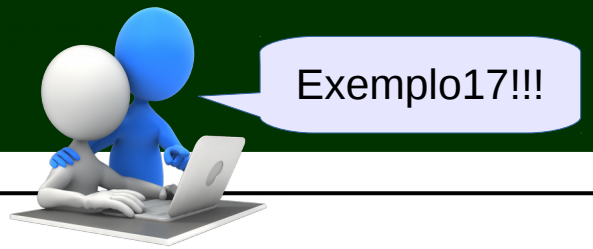
Posso colocar uma nota na posição 10?

Teste aí!

Situação problema 2

- Cliente deseja implementar um sistema para controle de uma fila bancária para atendimento dos clientes.
  - O sistema de controle de fila de um banco opera da seguinte forma:
    - O cliente que chega a agência retira uma senha que pode ter prioridade preferencial (para gestantes, portadores de necessidades especiais e maiores de 65 anos) ou não preferencial.
    - A agência dispõe de um caixa preferencial e vários caixas não preferenciais. Ao decidir qual o próximo cliente a ser atendido, são consideradas duas políticas:
      - O caixa preferencial seleciona o primeiro cliente preferencial da fila. Não havendo clientes preferenciais, é selecionado o primeiro cliente da fila;
      - O caixa não preferencial respeita a ordem de chegada e seleciona o primeiro cliente que está aguardando na fila, preferencial ou não.






```java
package model;
import java.util.ArrayList;
public class FilaAtendimentoBancario<T> implements Fila<T>, FilaComPrioridade<T> {

  private ArrayList<T> elementos;

  public FilaAtendimentoBancario() {
    elementos = new ArrayList<>();
  }
  @Override
  public boolean enqueue(T c) {
    elementos.add(c);
    return true;
  }
  @Override
  public T dequeue() {
    T retorno = null;
    if(!elementos.isEmpty()){
      retorno = elementos.get(0);
      elementos.remove(retorno);
    }
    return retorno;
  }
  @Override
  public boolean isFull() {
    return false;
  }
  @Override
  public boolean isEmpty() {
    return elementos.isEmpty();
  }
```

```java
  @Override
  public T dequeuePrioritario() {
    T retorno = null;
    for(int i=0; i<elementos.size() && retorno == null; i++){
      Cliente c = (Cliente) elementos.get(i);
      if(c.isPreferencial()){
        retorno = elementos.get(i);
        elementos.remove(i);
      }
    }
    return retorno;
  }
}
```

O ArrayList passa a ser o único atributo necessário.

```
package model;
import java.util.ArrayList;
public class FilaAtendimentoBancario<T> implements Fila<T>, FilaComPrioridade<T> {

  private ArrayList<T> elementos;

  public FilaAtendimentoBancario() {
    elementos = new ArrayList<>();
  }
  @Override
  public boolean enqueue(T c) {
    elementos.add(c);
    return true;
  }
  @Override
  public T dequeue() {
    T retorno = null;
    if(!elementos.isEmpty()){
      retorno = elementos.get(0);
      elementos.remove(retorno);
    }
    return retorno;
  }
  @Override
  public boolean isFull() {
    return false;
  }
  @Override
  public boolean isEmpty() {
    return elementos.isEmpty();
  }
```

O add() garante que o novo elemento será inserido no final do ArrayList.

Recupera-se o elemento na posição zero do ArrayList e em seguida remove-se o objeto.

Faz-se uma busca por um cliente prioritário, se existir remove aquela posição da lista, independentemente de onde esteja.

Não quebra o TAD!

```
  @Override
  public T dequeuePrioritario() {
    T retorno = null;
    for(int i=0; i<elementos.size() && retorno == null; i++){
      Cliente c = (Cliente) elementos.get(i);
      if(c.isPreferencial()){
        retorno = elementos.get(i);
        elementos.remove(i);
      }
    }
    return retorno;
  }
}
```

Trabalhando

- **Exercício de Fixação** (antes do café)
  - 13

Doctor Lecter, não entendo uma coisa.

Se ArrayList é tão bom assim, facilita minha vida, por que aquele professor não deixou eu usar a tecnologia?

Será que ele não sabe procurar algo na internet? Não conhece startpage.com ou duckduckgo.com para se atualizar?

Uma maravilha de Java e o cara fica obrigando usar array!!!

Will, está com fome?

Como funciona um ArrayList

- Este tipo de lista é implementado como um Array que é dimensionado dinamicamente, ou seja, sempre que é necessário o seu tamanho aumenta em 50% do tamanho da lista.
- Significa que se você tiver uma lista de tamanho igual a 10 e ela “encher”, seu tamanho aumentará para 15 automaticamente.
- O custo para aumentar o tamanho de um ArrayList é alto, pois é feita uma cópia do array atual para um novo array com um novo tamanho.
  - Imagine um array com 10mil elementos que será copiado para um novo array para criação de mais 5 mil elementos? De fato é um alto custo.
- É altamente aconselhável que você já inicie seu Array com uma quantidade de elementos que atenda ao seu objetivo atual, sem a necessidade de criação dinâmica de novos espaços, ou seja, se você souber que terá que armazenar de 300 a 400 objetos em um Array, defina 500, pois é melhor sobrar espaço do que utilizar recurso do processador sem necessidade.

Exemplo

```
public Estudante(int prontuario, String nome) {
  this.prontuario = prontuario;
  this.nome = nome;
  notas = new ArrayList<>(4);
}
```

Ainda é vantagem usar ArrayList !

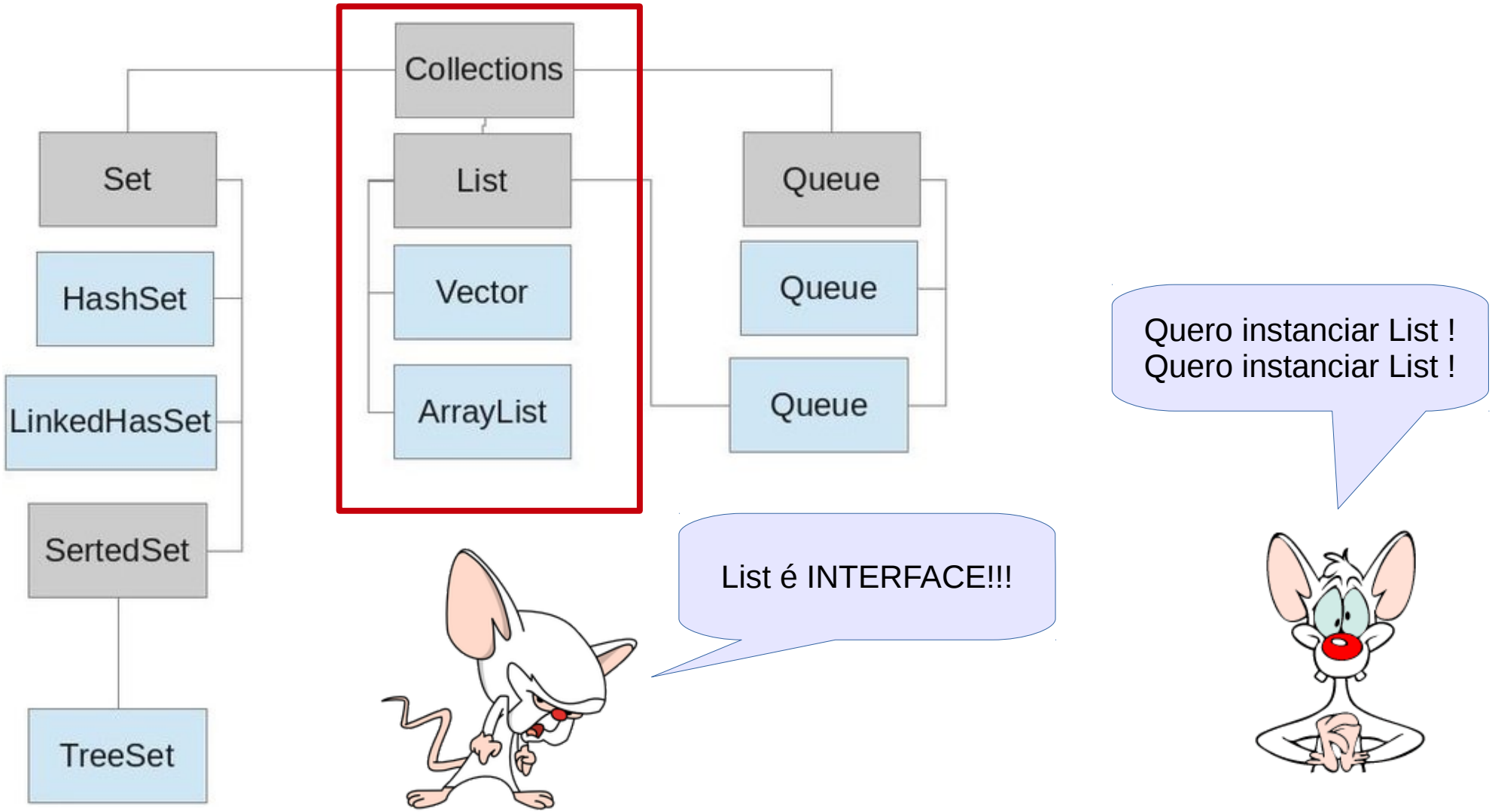
Existe alguma outra restrição?

Alternativas?

Não é adequado para variáveis de tipo primitivo (int, boolean, double). Sempre que é usamos existe uma conversão para um objeto equivalente. O programador não vê, mas o curso também é alto.

ArrayList não é sincronizado, quando estudar Multi-Thread entenderemos melhor isso.

API Java Collection



Vector

- A classe Vector é muito similar a ArrayList, porem é preciso esta atendo em algumas diferenças entre Ambas:
  - Vector é sincronizada (Podemos implementa Thread-Safe utilizando-a).
  - ArrayList cresce automaticamente nossa lista em 50%, já classe Vector aumenta o dobro, ou seja se temos uma lista com tamanho 20 utilizando Vector e caso queiramos aumentá-la esta lista vai fica com tamanho 40.

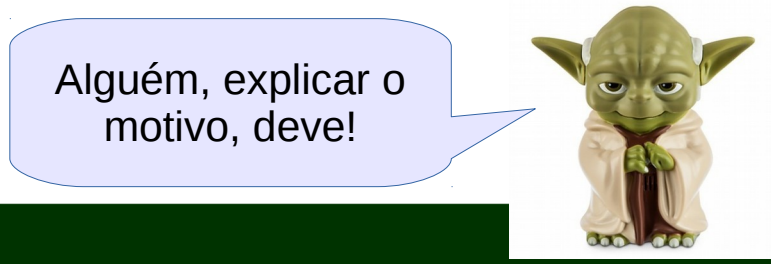
Vector

```java
public class Estudante {
  private int prontuario;
  private String nome;

  private Vector<Nota> notas;

  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;

    notas = new Vector<>(4);

  }
  public void limparNotas(){
    notas.clear();
  }
  public void addNota(double valor, double porcentagem){
    notas.add(new Nota(valor, porcentagem));
  }
  public void addNota(int nota, double valor, double porcentagem){
    notas.add(nota, new Nota(valor, porcentagem));
  }
```

```java
@Override
  public String toString(){
    StringBuilder sb = new StringBuilder();
    sb.append("Aluno: ");
    sb.append(getNome());
    sb.append("\nProntuário: ");
    sb.append(getProntuario());
    sb.append("\nNotas:");
    for (Nota n:notas) {
      sb.append("\nValor: ");
      sb.append(n.getNota());
      sb.append("\tPorcentagem: ");
      sb.append(n.getPorcentagem());
      sb.append("%");
    }
    return sb.toString();
  }
  public double media(){
    double soma=0;
    for(Nota n : notas){
      soma += n.parcial();
    }
    return soma;
  }...}
```

LinkedList

- Implementa uma “double linked list”, ou seja, uma lista duplamente encadeada.
- A sua principal diferença entre o ArrayList é na performance entre os métodos add, remove, get e set.
- Possui melhor performance nos métodos add e remove, do que os métodos add e remove do ArrayList, em compensação seus métodos get e set possuem uma performance pior do que os do ArrayList.
- Vamos fazer uma comparação entre a complexidade apresentada de cada método do ArrayList e o da LinkedList.
  - get(int index): LinkedList possui O(n) e ArrayList possui O(1)
  - add(E element): LinkedList possui O(1) e ArrayList possui O(n) no pior caso, visto que o array será redimensionado e copiado para um novo array.
  - add(int index, E element): LinkedList possui O(n) e ArrayList possui O(n) no pior caso
  - remove(int index): LinkedList possui O(n) e ArrayList possui O(n-index), se remover o último elemento então fica O(1)

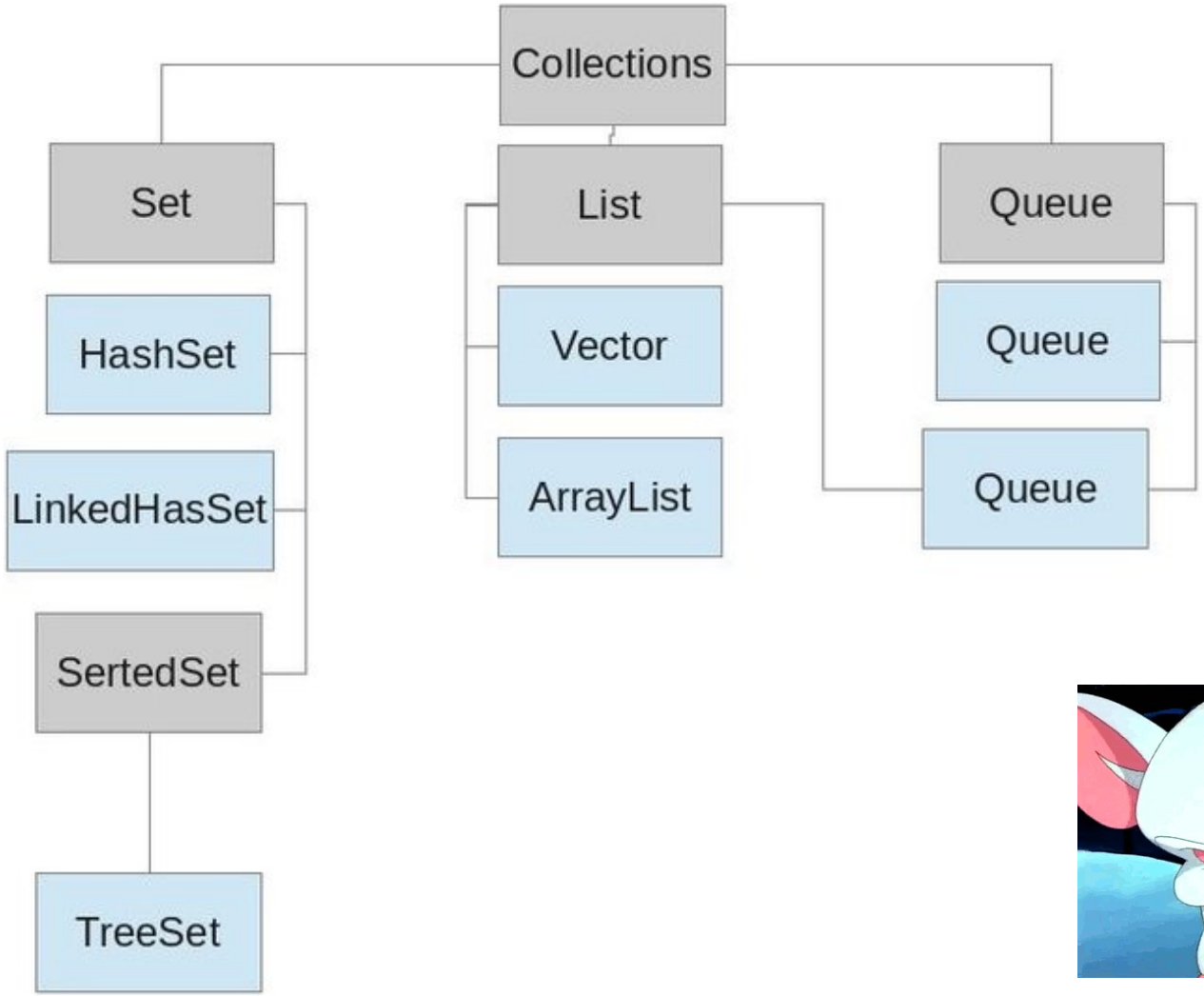


LinkedList

```
public class Estudante {
  private int prontuario;
  private String nome;
  private LinkedList<Nota> notas;
  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;

    notas = new LinkedList<>();

  }
  public void limparNotas(){
    notas.clear();
  }
  public void addNota(double valor, double porcentagem){
    notas.add(new Nota(valor, porcentagem));
  }
  public void addNota(int nota, double valor, double porcentagem){
    notas.add(nota, new Nota(valor, porcentagem));
  }
```

Única diferença é que não se define o tamanho inicial. Não teria sentido fazer isso para uma lista encadeada.

API Java Collection



```java
public class Estudante {
  private int prontuario;
  private String nome;
  private List<Nota> notas;
  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new ArrayList<>(4);
  }
```

```java
public class Estudante {
  private int prontuario;
  private String nome;
  private List<Nota> notas;
  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new Vector<>(4);
  }
```

```java
public class Estudante {
  private int prontuario;
  private String nome;
  private List<Nota> notas;
  public Estudante(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    notas = new LinkedList<>();
  }
```

Agora que sei **POLIMORFISMO,** entendo que List é apenas uma **INTERFACE** e conheço as **CLASSES CONCRETAS** que implementam a interface List posso dominar o mundo!

Agora quero usar Array!

Trabalhando

- **Exercício de Fixação**
  - 14

Material Adicional

- **Varargs**
  - https://www.devmedia.com.br/listas-de-argumentos-de-comprimento-variavel-em-java/25559
  - https://www.youtube.com/watch?v=vIthjvYNf08
- **ArrayList**
  - https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
  - https://www.devmedia.com.br/visao-geral-da-interface-collection-em-java/25822
  - https://www.devmedia.com.br/api-collections-em-java-fundamentos-e-implementacao-basica/28445
  - https://www.devmedia.com.br/diferenca-entre-arraylist-vector-e-linkedlist-em-java/29162
  - https://digaotutoriais.wordpress.com/2016/04/11/interface-set-list-map-e-queue/