Aula 3

Programação Orientada a Objetos

Prof. Leonardo Gomes

1

Conversa Inicial

2

- Visibilidade
- Encapsulamento
- Collections
- Iterator
- Classe localdate

3

Visibilidade

4

Visibilidade

- Private
- Protected
- Public
- Default

5

Visibilidade

Visibilidade	Public	Protected	Default	Private
Mesma classe	SIM	SIM	SIM	SIM
Classe qualquer no mesmo pacote	SIM	SIM	SIM	NÃO
Classe filha e mesmo pacote	SIM	SIM	SIM	NÃO
Classe filha e pacotes diferentes	SIM	SIM	NÃO	NÃO
Classe qualquer em outro pacote	SIM	NÃO	NÃO	NÃO

6

Visibilidade código

```
public class Aluno {
   private int matricula;
   protected int notas[];
   public String cpf;
   public String nome;

   public Aluno(String nome, String cpf){
      this.nome = nome;
      this.cpf = cpf;
      this.matricula = Cadastro.gerarNovaMatricula();
   }
   public void cadastrarNotas(){
      //codigo cadastro
   }
   public int mediaNotas(){
      //codigo media
   }

}
```

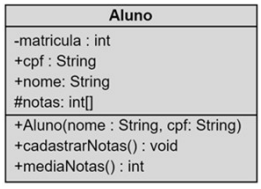
7

Visibilidade UML

Símbolo	Significado
+	Público
-	Privado
#	Protegido

8

Classe aluno



9

Encapsulamento

10

- **Um dos pilares da orientação a objetos**
- **Cápsula**
- **Protege o que está dentro**
- **Protege o que está fora**

11

- **Exemplo: monitor de um computador**
  - **Invólucro de plástico**
  - **Protege os componentes internos e os usuários**

12

Vantagens

- **Abstração oferecida: o funcionamento interno dos objetos da classe não fica visível ao programador que a utiliza**
- **Possibilidade de acrescentar funcionalidades à classe, desde que respeitando a interface original: isso manterá o sistema funcional sem alterações**

13

- **Simplificação da utilização dos objetos em um alto nível: acelera o desenvolvimento dos códigos**
- **O sistema fica robusto a mudanças internas: mesmo que existam mudanças no código dos métodos, a integração com o sistema será garantida desde que se respeite a interface**

14

Métodos set e get

```
public class Horario {
      private int hora;
      private int minuto;
      private int segundo;

      public void setHora(int h) {
           if (h > 23 || d < 0) {
               System.out.println("Valor inválido");
           }else {
               hora = h;
           }
      }

      public int getHora() {
          return hora;
      }
}
```

15

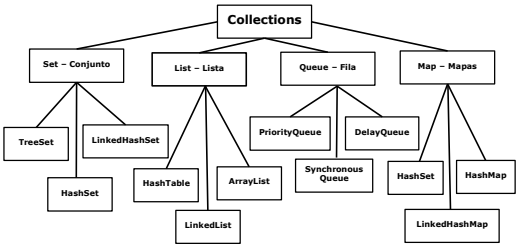
Collections

16

- **Importante API**
- **Implementam diferentes estruturas de dados**
- **Encapsuladas seguindo interfaces comuns**

17

Collections



18

Listas

```
ArrayList<String> pessoas = new ArrayList<String>();
//LinkedList<String> pessoas = new LinkedList<String>();

pessoas.add("Mario"); //Adição de novos elementos
pessoas.add("Luigi");
pessoas.add("Peach");
pessoas.add("Yoshi");
System.out.println(pessoas);//Lista dos elementos
item1 = pessoas.get(0); //retorna o elemento de índice 0
pessoas.remove(3);//remove o elemento de índice 3
total = pessoas.size();//retorna a quantidade de elementos
pessoas.clear(); //Remove todos os elementos
```

19

HashMap

```
HashMap<String, String> capitais = new HashMap<String, String>();
capitais.put("Brasil", "Brasília");
capitais.put("Argentina", "Buenos Aires");
capitais.put("Paraguai", "Assunção");
capitais.put("Uruguai", "Montevidéu");
System.out.println(capitais);//Imprimindo tudo
System.out.println(capitais.get("Uruguai"));//Imprimindo apenas a
capital do Uruguai
```

20

Métodos

- **sort(List)**
- **shuffle(List, Random)**
- **max(Collection,Comparator)**
- **min(Collection,Comparator)**
- **reverse(List)**

21

Iterator

22

Iterator

- **Utilizado para navegar entre os dados**
- **Funcionamento semelhante aos ponteiros**
- **No entanto, carrega diversas informações**
- **Funciona para diferentes estruturas de dados**

23

Exemplo Iterator

```
ArrayList<Integer> lista = new ArrayList();
HashSet<Integer> conjunto = new HashSet<Integer>();
HashMap<String,Integer> mapa = new
HashMap<String,Integer>();
int soma;

soma=0;//For simples
for(int i=0;i<lista.size();i++) {
      soma += lista.get(i);
}

soma=0;//For each
for(int item : lista) {
      soma += item;
}

soma=0;//Iterator
//Iterator it = mapa.entrySet().iterator();
//Iterator it = conjunto.iterator();
Iterator it = lista.iterator();
while(it.hasNext()) {
      soma += (int)it.next();
}
```

24

Classe Localdate

25

LocalDate

- **Forma robusta de representação de datas**
- **Substitui java.utilData e java.util.Calendar**
- **Versão 8 do Java em diante**

26

Exemplo

```
01.  public static void main(String[] args) {
02.
03.      //Captura a data de hoje
04.      LocalDate dataHoje = LocalDate.now();
05.
06.      System.out.println("Original: " + dataHoje);
07.      DateTimeFormatter formatador =
   DateTimeFormatter.ofPattern("dd/MM/yyyy");
08.      String dataForm= hoje.format(formatador);
09.      System.out.println("Formatado : " + dataForm);
10.
11.  }
```

27

Formato

- **dd = dia do mês em dois dígitos**
- **MM = mês em dois dígitos**
- **yyyy = ano em quatro dígitos**
- **HH = horas, até 23, em dois dígitos**
- **mm = minutos em dois dígitos**
- **ss = segundos em dois dígitos**
- **hh = horas, até 12, em dois dígitos**

28

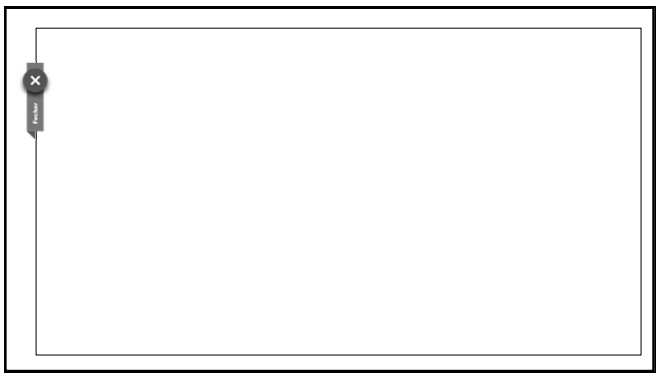
- **d = dia do mês em um ou dois dígitos**
- **M = mês do ano em um ou dois dígitos**
- **yy = ano em dois dígitos**
- **H = horas, até 23, em um ou dois dígitos**
- **m = minutos em um ou dois dígitos**
- **s = segundos em um ou dois dígitos**
- **h: horas, até 12, em um ou dois dígitos**

29

Exemplo hora

```
01.  public static void main(String[] args) {
02.    //Obtém LocalDateTime trazendo o horário atual
03.    LocalDateTime horario = LocalDateTime.now();
04.
05.    System.out.println("LocalDateTime antes: " + horario);
06.    DateTimeFormatter formatador =
   DateTimeFormatter.ofPattern("HH:mm:ss");
07.    String horarioFormatado = agora.format(formatador);
08.    System.out.println("LocalDateTime depois: " +
   horarioFormatado);
09.
10.  }
```

30



31