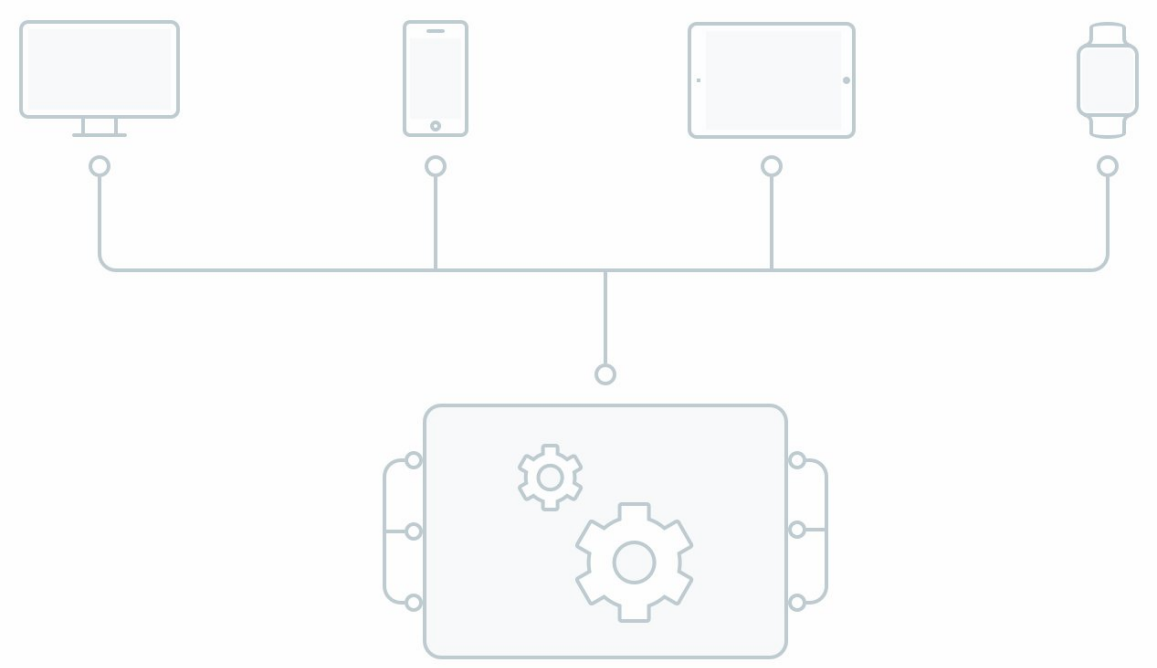


Aula 11

Interfaces

Rafael Geraldeli Rossi

Introdução

- **Interface (ciência da computação)** [Wikipedia, 2013]:
  - "*Uma interface, em ciência da computação, é a fronteira que define a forma de comunicação entre duas entidades*"
  - "*Ela pode ser entendida como uma abstração que estabelece a forma de interação da entidade com o mundo exterior, através da separação dos métodos de comunicação externa dos detalhes internos da operação, permitindo que esta entidade seja modificada sem afetar as entidades externas que interagem com ela*"

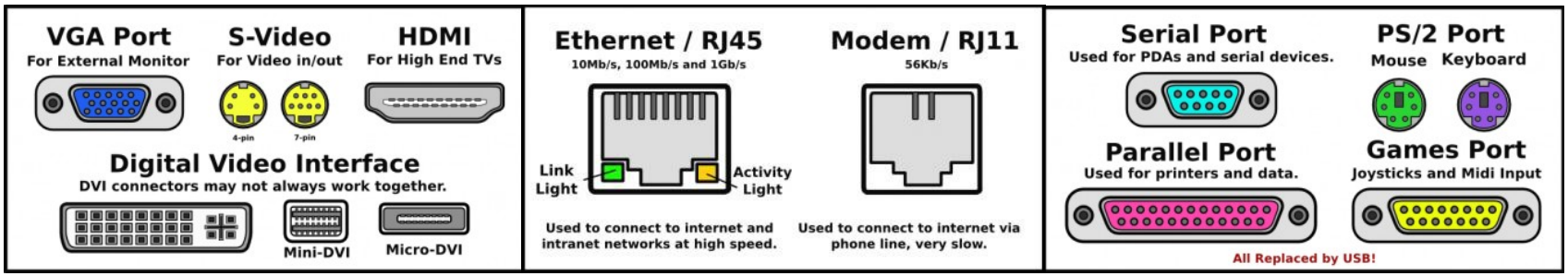
Introdução

- As interfaces mais faladas na computação são:
  - **Interface gráfica**: meio com que o usuário pode interagir com o programa por meio de botões, caixas de textos, listas, etc

https://s-media-cache-ak0.pinimg.com/736x/1a/1b/86/1a1b86a54c2c8e5d0c5ef6da59123ad4.jpg

Introdução

- As interfaces mais faladas na computação são:
  - **Interface física**: define o meio físico que será utilizados na comunicação entre dois computadores ou um computador e um periférico, além do tipo de informação que será trocada utilizando os meios físicos



http://kingorlin.com/files/custom/news/pcports/video-ports1.png

Introdução

- **Interface (POO)**: uma interface em programação orientada a objetos pode **definir uma série de métodos, mas nunca conter implementação deles → só expõe o que o objeto deve fazer, e não como ele faz, nem o que ele tem**

Introdução

Mas não dá pra fazer isso com métodos abstratos???

Resposta: Dá...

Mas então por que preciso aprender interfaces?

Respostas:

1. Uma classe não pode herdar de mais de uma classe (possível conflito de métodos)
2. Ao herdar de uma classe, herdamos todos os campos e implementações de métodos existentes
3. Além, disso, conceitualmente falando, ao herdar de uma classe, estamos especializando a classe herdada, isto é, queremos aprimorar uma classe existente

Introdução

- Porém, há situações em que apenas queremos **garantir que determinados métodos serão implementados** para que possamos realizar um **tratamento polimórfico**, sem precisar herdar atributos ou implementações existentes

- Além disso, podemos querer forçar que uma classe implemente **métodos que são especificados em mais de um lugar**

- Para isso, vamos utilizar as **INTERFACES**

Introdução

- A interface especifica quais implementações devem ser realizadas mas não especifica como essas implementações serão realizadas

- Além disso, a interface provê a capacidade de que **classes não relacionadas implementem um conjunto de métodos comuns** $\rightarrow$ forma útil para definir funcionalidades/comportamento comum para classes não relacionadas

Introdução

- Com isso, classes correspondentes à uma Pessoa, um Carro, uma ContaBancaria, etc, que possuem campos bem diferentes, podem ser tratadas polimorficamente $\rightarrow$ classes que implementam as mesmas interfaces terão as mesmas chamadas de métodos
- Uma interface costuma ser utilizada no lugar de uma classe abstrata quando não há nenhuma implementação padrão a herdar
- A interfaces devem do tipo `public`

Introdução

- Uma interface `public` deve ser declarada em um arquivo com o mesmo nome da interface e a extensão de arquivo `.java`
- A Interface é muito utilizada em grandes projetos para obrigar o programador a seguir o padrão do projeto
- Por se tratar de um contrato onde o mesmo é obrigado a implementar seus métodos, ele deverá sempre seguir o padrão de implementação da Interface

Declarando Interfaces

- Uma **declaração de interface** inicia com a **palavra-chave interface** e contém **somente constantes e métodos** `abstract`

```
1   public interface PlayerMusica {
2
3       public abstract void trocarMusica();
4
5       public abstract void tocarMusica();
6
7       public abstract void pausarMusica();
8
9       public abstract void pararMusica();
10
11      public abstract void aumentarVolume();
12
13      public abstract void diminuirVolume();
14
15  }
```

Declarando Interfaces

- Diferentemente das classes, **todos os membros de interface devem ser** `public` e as interfaces **não podem especificar nenhum detalhe de implementação** como **declaração de métodos concretos** e **variáveis de instância**

- Todos os métodos declarados em uma interface são implicitamente métodos `public abstract`

- Todos os campos são implicitamente `public`, `static` e `final`

Utilizando Interfaces

- Para utilizar uma interface, uma classe concreta deve especificar que ela **implementa** a interface
- Deve-se implementar cada método abstrato definido na interface
- Implementar uma interface é como assinar um contrato com o compilar que afirma "Irei declarar todos os métodos especificados pela interface ou irei declarar minha classe `abstract`"

Utilizando Interfaces

- **OBSERVAÇÃO:** Uma classe que não implementa todos os métodos da interface é uma classe abstrata e deve ser declarada como `abstract`

- **OBSERVAÇÃO:** **o Java não permite que subclasses estendam mais de uma superclasse, mas permite que uma classe estenda uma superclasse e implemente quantas interfaces ela precisar**

Utilizando Interfaces

- Para especificar que uma classe implementa uma interface, adiciona a palavra-chave `implements` e o nome da interface no fim da primeira linha da declaração da classe

```
1  public class MP3Player implements PlayerMusica{
2
3  }
```

- Para implementar uma classe concreta, deve-se implementar todos os métodos métodos especificados na interface
- Pode-se implementar varias interfaces → Separar os nomes das interfaces por vírgula

```
public class MP3Player implements PlayerMusica, Comparable
```

Utilizando interfaces

```java
public class MP3Player implements PlayerMusica{
    String[] lista;
    int posMusica;
    int volume;

    public MP3Player(String lista){
        this.lista = lista.split(",");
        posMusica = 0;
        volume = 1;
    }

    @Override
    public void trocarMusica() {
        if(lista.length > 0){
            posMusica = (posMusica + 1) % lista.length;
            System.out.println("MP3: música trocada para " + lista[posMusica]);
            tocarMusica();
        }
    }
    @Override
    public void tocarMusica() {
        if(lista.length > 0){
            System.out.println("MP3: a música " + lista[posMusica] + " está tocando");
        }
    }
    @Override
    public void pausarMusica() {
        if(lista.length > 0){
            System.out.println("MP3: a música " + lista[posMusica] + " está pausada");
        }
    }
    @Override
    public void pararMusica() {
        if(lista.length > 0){
            System.out.println("MP3: a música " + lista[posMusica] + " está  parada");
        }
    }
    @Override
    public void aumentarVolume() {
        if(volume < 10){
            volume++;
            imprimeVolume();
        }
    }
    @Override
    public void diminuirVolume() {
        if(volume > 0){
            volume--;
            imprimeVolume();
        }
    }
    private void imprimeVolume(){
        String barra = "";
        for(int i=0;i<volume;i++){
            barra += "|";
        }
        System.out.println("MP3: volume está em " + barra);
    }
}
```

Utilizando interfaces

```java
import java.util.ArrayList;
import java.util.Random;

public class Celular implements PlayerMusica{

    ArrayList<String> playList;
    double nivelVolume;
    double bateria;
    int idMusica;
    Random rand;

    public Celular(ArrayList<String> playList){
        this.playList = playList;
        nivelVolume = 10;
        bateria = 100;
        idMusica = 0;
        rand = new Random();
    }

    @Override
    public void trocarMusica() {
        if(playList.size() > 0 && bateria > 0){
            idMusica = rand.nextInt(playList.size());
            System.out.println("Celular: música trocada para " + playList.get(idMusica));
            tocarMusica();
            bateria -= 1.5;
        }
    }
    @Override
    public void tocarMusica() {
        if(playList.size() > 0){
            System.out.println("Celular: a música " + playList.get(idMusica) + " está tocando");
        }
    }
    @Override
    public void pausarMusica() {
        if(playList.size() > 0){
            System.out.println("Celular: a música " + playList.get(idMusica) + " está pausada");
        }
    }
    @Override
    public void pararMusica() {
        if(playList.size() > 0){
            System.out.println("Celular: a música " + playList.get(idMusica) + " está  parada");
        }
    }
    @Override
    public void aumentarVolume() {
        if(nivelVolume < 100){
            nivelVolume += 0.1;
            exibirVolume();
        }
    }
    @Override
    public void diminuirVolume() {
        if(nivelVolume > 0){
            nivelVolume -= 0.1;
            exibirVolume();
        }
    }
    private void exibirVolume(){
        System.out.println("Celular: volume está em " + nivelVolume);
    }
```

Utilizando interfaces

```
4   public class Programa {
5
6       public static void main(String[] args){
7
8           ArrayList<PlayerMusica> players = new ArrayList<>();
9           String lista1 = "Pradise,Clock,The Scientist,In My Place,A Sky Full of Stars";
10          players.add(new MP3Player(lista1));
11
12          ArrayList<String> lista2 = new ArrayList<String>();
13          lista2.add("Nosso santo bateu");
14          lista2.add("10%");
15          lista2.add("Você partiu meu coração");
16          lista2.add("Despacito");
17          players.add(new Celular(lista2));
18
19          boolean sair = false;
20          while(sair == false){
21              int dispositivo = exibeMenuOpcoes(players.size());
22              if(dispositivo == -1 || dispositivo == 0){
23                  sair = true;
24              }else{
25                  int op = exibeMenuAcoes();
26                  if(op == -1){
27                      sair = true;
28                  }else if(op == 1){
29                      players.get(dispositivo -1).tocarMusica();
30                  }else if(op == 2){
31                      players.get(dispositivo -1).pausarMusica();
32                  }else if(op == 3){
33                      players.get(dispositivo -1).pararMusica();
34                  }else if(op == 4){
35                      players.get(dispositivo -1).trocarMusica();
36                  }else if(op == 5){
37                      players.get(dispositivo -1).aumentarVolume();
38                  }else if(op == 6){
39                      players.get(dispositivo -1).diminuirVolume();
40                  }
41              }
42          }
43
44      }
45      public static int exibeMenuAcoes(){
46          int op = Integer.parseInt(JOptionPane.showInputDialog(null, "1 - Tocar\n2 - Pausar\n3 - Parar\n4 - Trocar\n5 - Aumentar Volume\n6- - Diminuir Volume"));
47          if(op >= 1 && op <=6){
48              return op;
49          }
50          JOptionPane.showMessageDialog(null, "Opção Inválida");
51          return -1;
52      }
53      public static int exibeMenuOpcoes(int numOps){
54          int op = Integer.parseInt(JOptionPane.showInputDialog(null, "Digite 0 para sair ou de 1 a " + numOps + " para selecionar o dispositivo correspondente"));
55          if(op >= 0 && op <=numOps){
56              return op;
57          }
58          JOptionPane.showMessageDialog(null, "Opção Inválida");
59          return -1;
60      }
```

Ex: Interface Pagavel

```
1   public interface Pagavel {
2
3       public double calculaPagamento();
4
5   }
```

Ex: Interface Pagavel

```
8   public class FaturaConta implements Pagavel{
9
10      private String numeroNota;
11      private String descricaoNota;
12      private int qtd;
13      private double precoItem;
14
15      public FaturaConta(String numeroNota, String descricaoNota, int qtd, double precoItem){
16          this.numeroNota = numeroNota;
17          this.descricaoNota = descricaoNota;
18          this.qtd = qtd;
19          this.precoItem = precoItem;
20      }
21
22      @Override
        public double calcularQtdPagamento(){
24          return this.getQtd() * this.getPrecoItem();
25      }
26
27      @Override
        public String toString() {
29          return "FaturaConta{" + "numeroNota=" + numeroNota + ", descricaoNota=" + descricaoNota +
30      }
31
32      public String getNumeroNota() {
33          return numeroNota;
```

Ex: Interface Pagavel

```
12  public abstract class Empregado implements Pagavel {
13
14      String primeiroNome;
15      String segundoSome;
16      String CPF;
17
18      public Empregado(String primeiroNome, String segundoSome, String CPF) {
19          this.primeiroNome = primeiroNome;
20          this.segundoSome = segundoSome;
21          this.CPF = CPF;
22      }
23
24      @Override
        public abstract double calcularQtdPagamento();
26
27      @Override
        public String toString() {
29          return "Empregado{" + "primeiroNome=" + primeiroNome + ", segundoSome=" + segundoSome + ", CPF=" + CPF + '}';
30      }
31
32      public String getPrimeiroNome() {
33          return primeiroNome;
34      }
35
36      public void setPrimeiroNome(String primeiroNome) {
```

Ex: Interface Pagavel

```
12  public class EmpregadoRemunerado extends Empregado{
13
14      private double salario;
15
16      public EmpregadoRemunerado(double salario, String primeiroNome, String segundoSome, String CPF) {
17          super(primeiroNome, segundoSome, CPF);
18          this.salario = salario;
19      }
20
21      @Override
        public double calcularQtdPagamento(){
23          return this.getSalario();
24      }
25
26      @Override
        public String toString() {
28          return "Empregado{" + "primeiroNome=" + primeiroNome + ", segundoSome=" + segundoSome + ", CPF="
29      }
30
31      public void setSalario(double salario) {
32          if(salario < 0){
33              this.salario = 0;
34          }else{
35              this.salario = salario;
36          }
```

Ex: Interface Pagavel

```
12  public class Principal {
13
14      public static void main(String[] args){
15
16          Pagavel[] objetosPagaveis = new Pagavel[4];
17
18          objetosPagaveis[0] = new FaturaConta("01234", "banco de couro", 2, 500.50);
19          objetosPagaveis[1] = new FaturaConta("75360", "picolé de pudim", 5, 2.50);
20          objetosPagaveis[2] = new EmpregadoRemunerado("Rafael", "Rossi", "000777888666", 1200.00);
21          objetosPagaveis[3] = new EmpregadoRemunerado("Ricardo", "Marcacini", "1716924007", 14000.00);
22
23          for(Pagavel objPagavel : objetosPagaveis){
24              System.out.println(objPagavel.toString() + " - " + objPagavel.calcularQtdPagamento());
25          }
26
27      }
28
29  }
```

Interfaces.Principal > main > for (Pagavel objPagavel : objetosPagaveis) >

Saída - Teste (run)

```
run:
FaturaConta{numeroNota=01234, descricaoNota=banco de couro, qtd=2, precoItem=500.5} - 1001.0
FaturaConta{numeroNota=75360, descricaoNota=picolé de pudim, qtd=5, precoItem=2.5} - 12.5
Empregado{primeiroNome=Rafael, segundoSome=Rossi, CPF=000777888666, Salário=1200.0} - 1200.0
Empregado{primeiroNome=Ricardo, segundoSome=Marcacini, CPF=1716924007, Salário=14000.0} - 14000.0
```

Campos e Métodos `Default` em Interfaces

- Pode-se declarar campos em uma interface
- Obrigatoriamente, todos os campos de uma interface serão **públicos**, **estáticos** e **finais**
- Portanto, é necessário a especificação do valor do campo no momento da sua declaração
- Pode-se também especificar uma implementação de um método em uma interface utilizando a palavra-chave `default`

Campos e Métodos `Default` em Interfaces

```
8   public interface TesteInterface {
9
10      String significado = "Esta interface só serve para imprimir alguma coisa na tela.";
11
12      public default void imprimeAlgo(){
13          System.out.println("Uhul!!!!!!");
14      }
15
16
17  }
```

Drivers

- O polimorfismo é particularmente eficaz para implementar os chamados sistemas de *software* em camadas
- Em sistemas operacionais, por exemplo, cada tipo de dispositivo físico opera diferentemente dos outros
- Mesmo assim, os comandos para ler ou gravar os dados de e a partir de dispositivos poderiam ter certa uniformidade
- Para cada dispositivo, o sistema operacional utiliza um *software* chamado *driver* de dispositivo para controlar toda a comunicação entre o sistema e o dispositivo

Drivers

- Um sistema orientado a objetos fornece uma "interface" apropriada para todos os *drivers* de dispositivos
- Com isso, todas as classes que implementa a interface irão apresentar a mesma forma de comunicação
- Os métodos de um *driver* de dispositivo são declarados como métodos abstratos na interface (ou superclasse abstrata)
- As implementações desses métodos abstratos são fornecidas nas subclasses que correspondem aos tipos de *drivers* de dispositivos específicos

Engenharia de Software

- Os engenheiros de software mais radicais dize que toda classe deve ser "interfaceada", isto é, só devemos nos referir a objetos através de suas interfaces
- Alguns autores famosos da área de engenharia de *software* dizem que "deve-se programar voltado a interface e não à implementação" (padrão de design)
- O pai da linguagem Java (James Gosling) fala que ao utilizar a interface, você está fazendo um "encapsulamento puro"
- **OBSERVAÇÃO:** o professor da disciplina acha isso tudo muito radical

Interface comuns da Java API

- Varias interfaces encontradas na Java API poderão ser usadas em suas aplicações
- Essas interfaces são implementadas e usadas da mesma maneira como as interfaces que você mesmo cria
- As interfaces da API do Java permitem que você use suas próprias classes dentre das estruturas fornecidas pelo Java, como comparar os objetos dos seus próprios tipos, criar e executar tarefas que ocorrem concorrentemente, ...

Interfaces Populares na Java API

Interface	**Descrição**
`Comparable`	O Java contém vários operadores de comparação (por exemplo, <. <=, >, >=, ==, !=) que permitem comparar valores primitivos. Contudo, esses operadores não podem ser utilizados para comparar objetos. A interface `Comparable` é utilizada para permitir que objetos de uma classe que implementar a interface sejam comparados entre si. A interface `Comparable` é comumente utilizada para ordenar objetos em uma coleção, como um *array*

Interfaces Populares na Java API

Interface	**Descrição**
`Serializable`	Essa interface é usada para identificar classes cujos objetos podem ser gravados (serializados) ou lidos (desserializados) em algum tipo de armazenados ou transmitidos por uma rede
`Runnable`	Implementada por qualquer classe por meio da qual objetos dessa classe devem ser capazes de executar em paralelo utilizando uma técnica chamada *multithreading*. A interface contém um método `run`, que descreve o comportamento de um objeto quando executado

Interfaces Populares na Java API

Interface	**Descrição**
Interfaces `listener` de eventos com GUIs	Gerenciadores (*handlers*) de eventos são declarados em classes que implementam uma interface ouvinte de eventos apropriada. Cada interface ouvinte de eventos especifica um ou mais métodos que devem ser implementados para responder a interações de usuário
`SwingConstants`	Contém um conjunto de constantes utilizadas em programas GUI para posicionar elementos GUI na tela

Extra: Ordenando Utilizando a Interface `Comparable`

- Comparação relativa entre tipos primitivos é direta $\rightarrow$ >, <, ...
- Mas e a comparação relativa entre objetos?
- Comparação relativa entre objetos são úteis quando, por exemplo, se quer ordenar uma coleção de objetos
- A classe `Collection` contém um método `sort` que realiza a ordenação de objetos desde que a classe dos respectivos objetos implemente a interface `Comparable` (pacote `java.util`)

Extra: Ordenando Utilizando a Interface `Comparable`

- Ao implementar a interface `Comparable`, é necessário implementar o método `compareTo(Object obj)`
- Deve-se definir qual ou quais campos de acordo com alguma ordem devem ser utilizados na comparação
- O método `compareTo` deve retornar
  - `-1` se o objeto atual é menor que um outro objeto
  - `+1` se o objeto atual é maior que um outro objeto
  - `0` se os objetos são iguais

Extra: Ordenando Utilizando a Interface Comparable

```
9   public class Conta implements Comparable<Conta>{
10
11      private int id;
12      private String titular;
13
14      public Conta(int id, String titular) {
15          this.id = id;
16          this.titular = titular;
17      }
18
19      @Override
        public int compareTo(Conta outraConta){
21          if(this.id < outraConta.id){
22              return -1;
23          }else if(this.id < outraConta.id){
24              return 1;
25          }else{
26              return 0;
27          }
28      }
29
30      @Override
        public String toString() {
32          return "Conta{" + "id=" + id + ", titular=" + titular + '}';
33      }
```

Extra: Ordenando Utilizando a Interface Comparable

```
12  public class Principal {
13
14      public static void main(String[] args){
15
16          ArrayList<Conta> contas = new ArrayList<Conta>();
17          contas.add( new Conta(42345, "Academia") );
18          contas.add( new Conta(1247, "Mercado"));
19          contas.add( new Conta(35897, "Combustivel") );
20
21          Collections.sort(contas);
22
23          for(int i=0;i<contas.size();i++){
24              System.out.println(contas.get(i).toString());
25          }
26      }
27
28  }
```

Saída - Teste (run)

```
run:
Conta{id=1247, titular=Mercado}
Conta{id=35897, titular=Combustivel}
Conta{id=42345, titular=Academia}
```

Exercício

- Incremente o exercício do Projeto Banco acrescentando a opção `Listar Contas` da `Área Gerencial`
- Nesta opção, deve-se listar todas as contas (nº conta, nome do titular, CPF e saldo)
- Porém, as contas devem ser listadas de maneira decrescente em relação ao campo saldo

Material Complementar

- Java Interface Tutorial with Example

  `https://www.youtube.com/watch?v=5Aef6vnAxR8`

- Interfaces

  `https:`

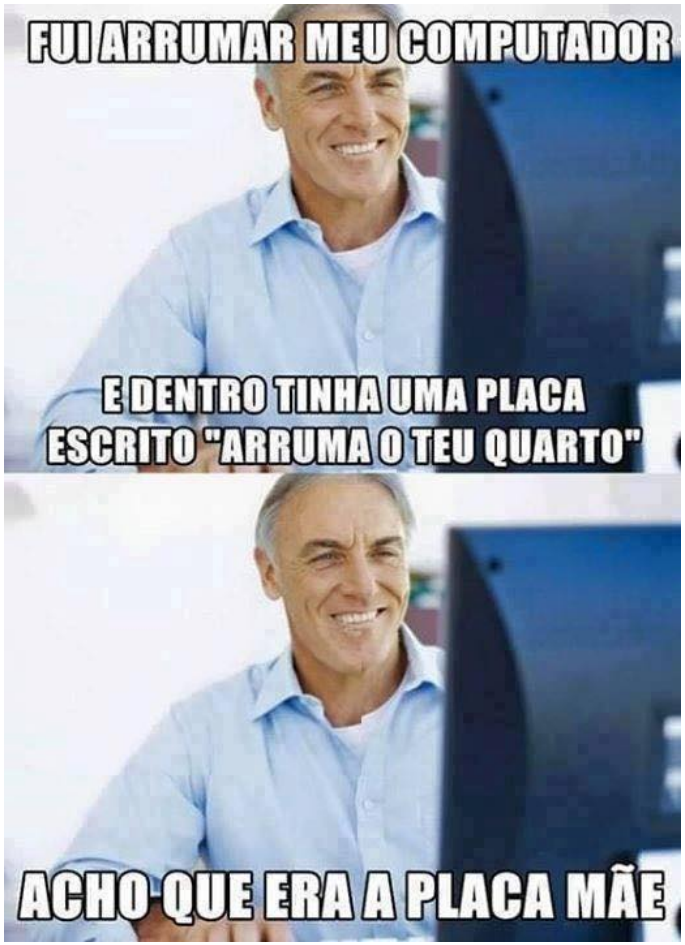
  `//www.caelum.com.br/apostila-java-orientacao-objetos/interfaces/`

- Ordenando coleções com Comparable e Comparator

  `http://blog.caelum.com.br/`

  `ordenando-colecoes-com-comparable-e-comparator/`

Imagem do Dia



Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I

Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.

Wikipedia (2013).
Inteface, `https://pt.wikipedia.org/wiki/Interface_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)`. Último acesso em 13 de junho de 2017.