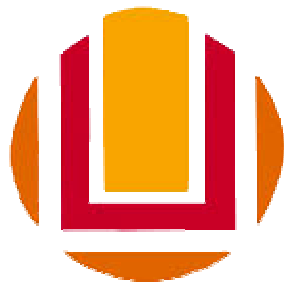


Minicurso Java

Fabio Aiub Sperotto

fabio.aiub@gmail.com





Orientação a objetos

- É uma forma de organização de código.
- Vantagens:
 - Facilita adição/alteração de funcionalidades.
 - Aumenta reuso de código.
 - Legibilidade de código.



Objetos

- Qualquer coisa é determinada por um objeto, mesas, cadeiras, produtos, veículos, datas, uma conta bancária, etc.
- Cada objeto possui estados e comportamentos.



Classes

- Classe representa um tipo de dado complexo.
- Classe especifica um objeto através de definições dos estados e comportamentos do mesmo.
- Objetos são instâncias das classes.
- Objetos possuem identidade única, ainda que possam compartilhar a mesma classe.



Exemplo

- Vamos pensar em veículos!



Classe Carro

```
public class Carro {  
  
    String modelo;  
    String cor;  
  
    public void ligar(){  
        System.out.println("Carro ligado");  
    }  
  
    public void desligar(){  
        System.out.println("Carro desligado");  
    }  
  
    public void acelerar(){  
        System.out.println("Carro acelerando, ao infinito e além!");  
    }  
  
    public void frear(){  
        System.out.println("Freaaaaaando.");  
    }  
  
    public void mudarMarcha(){  
        System.out.println("Marcha engatada");  
    }  
}
```

← Declaração da classe.

← Declaração dos atributos.

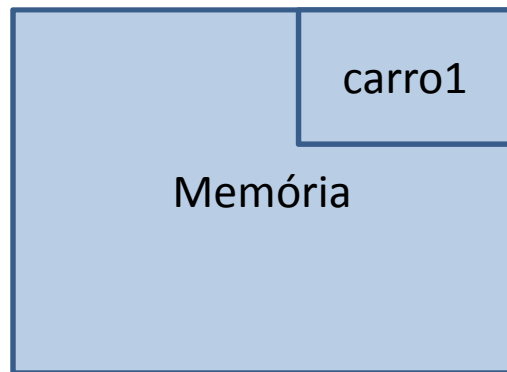
← Declaração dos métodos.

Arquivo será
Carro.java. Sempre o
nome da classe com a
extensão java.



Objeto da classe Carro

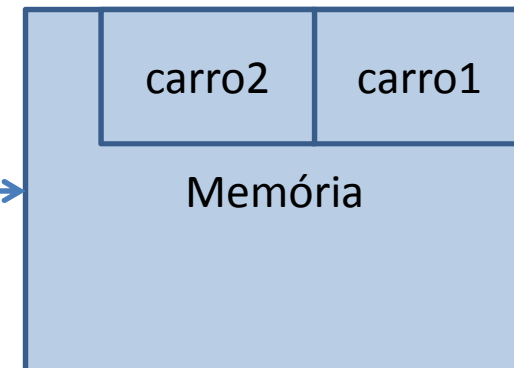
```
Carro carro1 = new Carro();
```



Instanciando um objeto da classe Carro chamado carro1. Utiliza a palavra reservada **new**.

```
Carro carro2 = new Carro();
```

Com um segundo objeto, agora temos carro1 e carro2:





Exemplo de objetos

```
public static void main(String args[]){  
  
    Carro carro1 = new Carro();  
    Carro carro2 = new Carro();  
  
    carro1.modelo = "fusca";  
    carro1.cor = "verde";  
  
    carro1.ligar();  
    carro1.mudarMarcha();  
    carro1.acelerar();  
    carro1.frear();  
}
```

Carro ligado
Marcha engatada
Carro acelerando, ao infinito e além!
Freaaaaando.



Herança

- Herdar estados e comportamentos de outras classes.
- O que é comum é compartilhado pela herança.
- Veículos -> carros, motos, aviões.



Interface

- É um componente que define um contrato de estados e comportamentos para as classes.
- Não define a programação de métodos, mas possui um conjunto de características que uma classe deve possuir.



Pacotes e APIs

- Organização de classes.
- Application Programming Interface.
- Bibliotecas de programação, da linguagem ou não, para utilizar no código-fonte.

! **package** -> define a qual pacote a classe se refere.
import -> define a importação de uma
• classe/biblioteca.



Modificadores de acesso (Classes)

- **public:** classes podem ser utilizadas por objetos de fora do pacote.
- **abstract:** não pode ter objetos instanciados.
- **final:** a classe não pode ter subclasses.



Modificadores de acesso (Métodos)

- **public:** o método pode ser acessado por qualquer classe em qualquer pacote.
- **private:** torna uma variável ou função acessível somente nas subclasses da classe ou nas classes do mesmo pacote.
- **protected:** torna uma classe, método ou variável acessível por qualquer outra classe.
- **abstract:** não implementa funcionalidade.



Modificadores de acesso (Métodos)

- **final**: o método não pode ser sobrescrito, sobreposto.
- **static**: método pode ser acessado diretamente pela sua classe (Classe.método).
- **native**: indica que um método é escrito em uma linguagem nativa como C ou C++.
- **synchronized**: indica que um método só pode ser acessado por uma única `*thread*` por vez.



Modificadores de acesso (Atributos)

- **public:** o atributo pode ser acessado por qualquer um.
- **protected:** o atributo pode ser acessado por subclasses da classe ou por classes no mesmo pacote.
- **private:** torna um atributo acessível somente dentro de sua própria classe.



Modificadores de acesso (Atributos)

- **final**: indica que o atributo guarda um valor fixo que não pode ser alterado.
- **static**: o atributo definido como static compartilha seu valor por todos os objetos da classe.



Exemplo modificadores

```
public class Carro {  
  
    private String modelo;  
    private String cor;  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
  
    public String getCor() {  
        return cor;  
    }  
  
    public void setCor(String cor) {  
        this.cor = cor;  
    }  
}
```



Exemplo com modificadores

```
public static void main(String args[]){  
  
    Carro carro1 = new Carro();  
    Carro carro2 = new Carro();  
  
    carro1.setModelo("fusca");  
    carro1.setCor("verde");  
  
    carro1.ligar();  
    System.out.println("Modelo: "+carro1.getModelo());  
    carro1.mudarMarcha();  
    carro1.acelerar();  
    carro1.frear();  
}
```



Métodos construtores

Sem construtor default:

```
public class Carro {  
  
    private String modelo;  
    private String cor;  
  
}
```

Com construtor default:

```
public class Carro {  
  
    private String modelo;  
    private String cor;  
  
    public Carro(){  
  
    }  
}
```

```
public class Carro {  
  
    private String modelo;  
    private String cor;  
  
    public Carro(){  
  
    }  
  
    public Carro(String modeloCarro){  
        this.modelo = modeloCarro;  
    }  
}
```

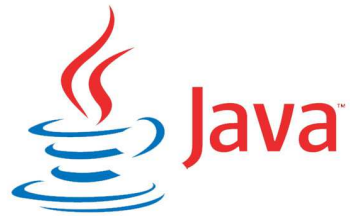
Construtor inicializando
atributo modelo.

! Construtores não são métodos,
são construtores.

Tipo especial de referência a métodos e variáveis

- **super**: referencia a variável imediatamente da super classe.
- **this**: referência a uma variável ou método da instância (objeto) corrente.





Minicurso Java – Aula 2



Apostila, slides e código-fonte base em

<https://github.com/fabiosperotto/minicursoJava>

O que será visto hoje:

- Arrays, Coleções.
- Tratamento de erros.
- Mais atalhos de produtividade (Eclipse).
- Exercícios.



Arrays

```
//arrays/matrices:
```

```
public double[] faixasFrequencia = new double[3];
```

```
public double faixasFrequencia[] = {98.2,99.0,100.1,101.5};
```

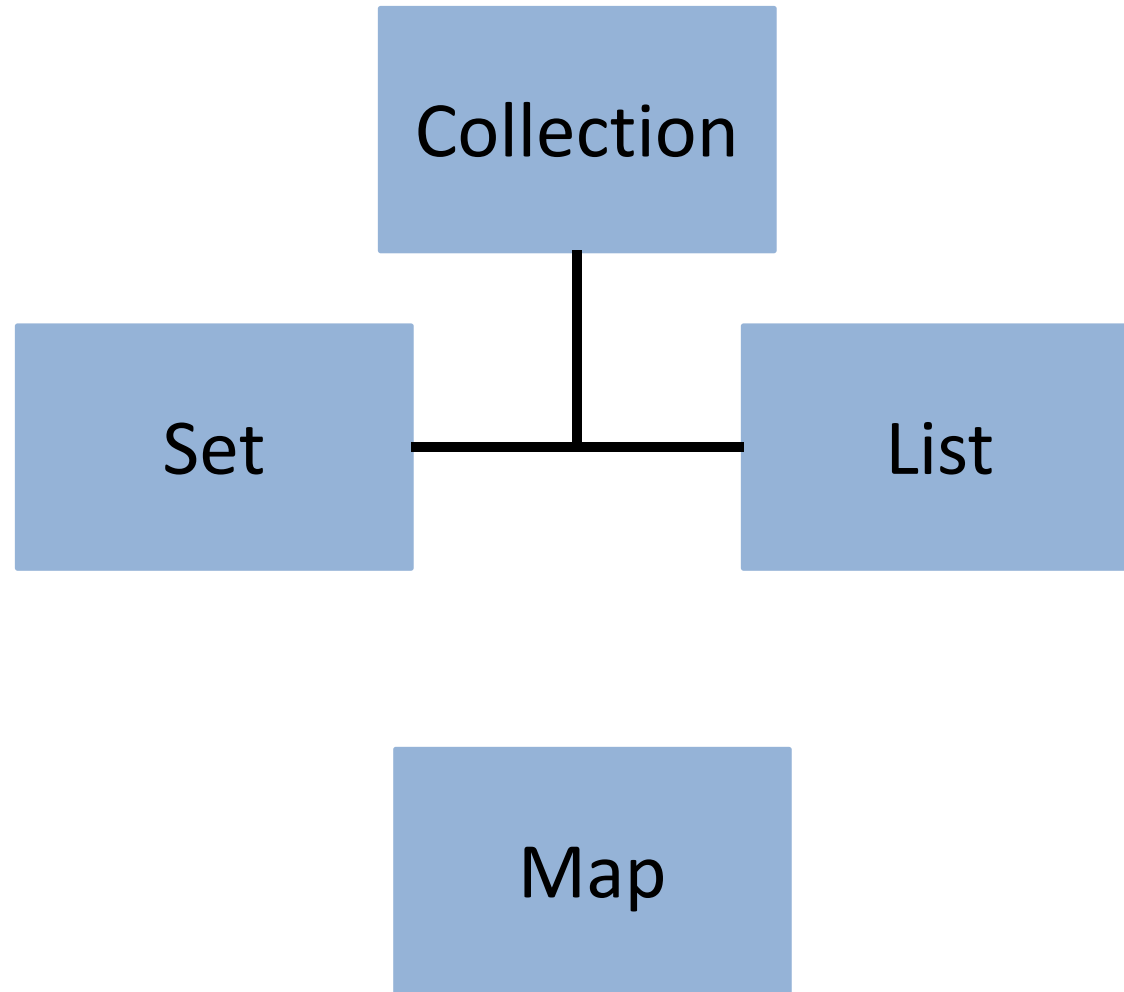
```
faixasFrequencia[0] = 98.2;
```

```
faixasFrequencia[0] = 99.0;
```

```
faixasFrequencia[0] = 100.1;
```

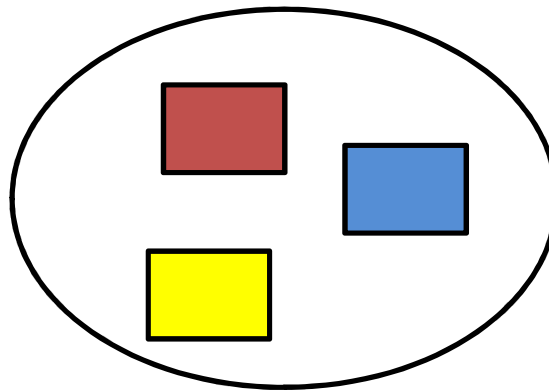


Coleções





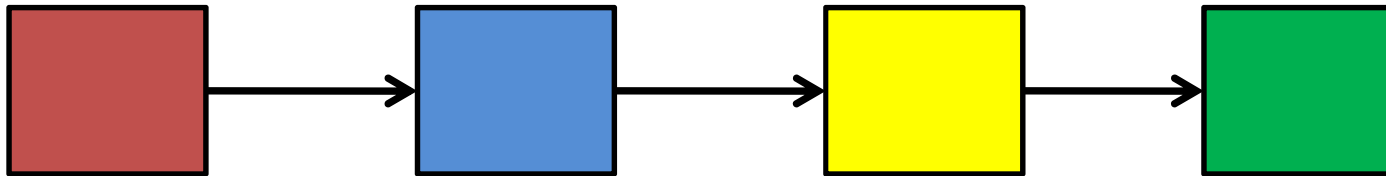
Coleções - Set



```
import java.util.HashSet;  
HashSet<Double> faixasFrequencia = new HashSet<Double>();  
faixasFrequencia.add(98.2);  
faixasFrequencia.add(99.0);  
faixasFrequencia.add(100.1);
```



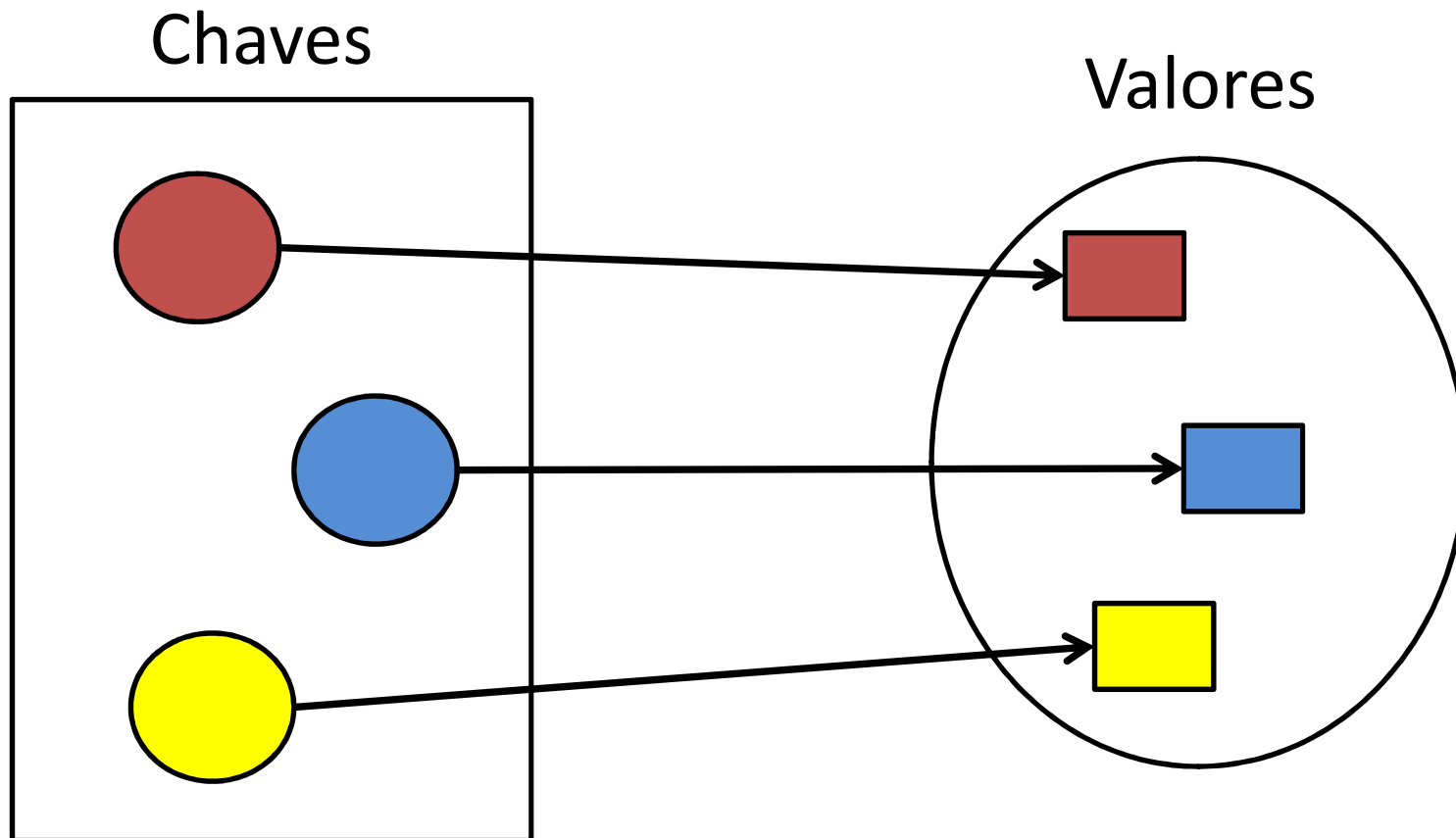

Coleções - List



```
import java.util.ArrayList;
public ArrayList faixasFrequencia = new ArrayList();
public ArrayList<Double> faixasFrequencia = new ArrayList<Double>();
faixasFrequencia.add(98.2);
faixasFrequencia.add(99.0);
faixasFrequencia.add(100.1);
```



Coleções - Map





Coleções - Map

```
import java.util.HashMap;  
HashMap<Integer,String> faixasCD = new HashMap<Integer,String>();  
faixasCD.put(1, "St. Anger");  
faixasCD.put(2, "Fuel");  
faixasCD.put(3, "Shoot Me Again");  
faixasCD.put(4, "Seek and Destroy");
```

```
this.faixasCD.get(2);
```

→ O que será retornado?



List - Carros

1. Crie um novo pacote chamado `extra.veiculos`.
2. Crie uma classe pública `Carro` com os atributos `modelo` e `cor` do tipo privado. Com dois construtores: vazio e inicializando os atributos. Inclua também o método `main()` para executarmos a aplicação.
3. Gerar os `getters` and `setters` dos respectivos atributos.
4. Crie os objetos `carro1`, `carro2` e `carro3`.
5. Instancie um `ArrayList` `carros` e insira os 3 carros na Lista.



Tratamento de Exceções

```
try{  
    //bloco para executar  
  
}catch(Exception erro){  
  
    //tratamento dos erros do bloco de execução aqui  
  
}catch(Error erro2){  
  
    //outro tratamento de erro  
  
}
```



Atalhos teclado (Eclipse)

- Sempre utilize ctrl+espaço para que o editor sugira código (muito útil para completar nomes de classes/métodos/variáveis).
- Syso ctrl+espaço: abreviatura para `System.out.println();`
- Alt+shift+S+R -> abre opção de geração de código *Getters and Setters*.
- Mais atalhos: ctrl+shift+L.

- Exercícios Apostila pág. 12 e 13.
- Códigos base disponíveis em <https://github.com/fabiosperotto/minicursoJava>, Fork o projeto ou baixe-o através do botão ZIP.



DÚVIDAS

A fatal exception 0E has occurred at 0028:C0011E36 in UXD UMM(01) + 00010E36. The current application will be terminated.

- * Press any key to terminate the current application.
- * Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _



Swing

- Interface gráfica de usuário, GUI.

javax.accessibility

javax.swing

javax.swing.border

javax.swing.colorchooser

javax.swing.event

javax.swing.filechooser

javax.swing.plaf

javax.swing.plaf.basic

javax.swing.plaf.metal

javax.swing.plaf.multi

javax.swing.plaf.synth

javax.swing.table

javax.swing.text

javax.swing.text.html

javax.swing.text.html.parser

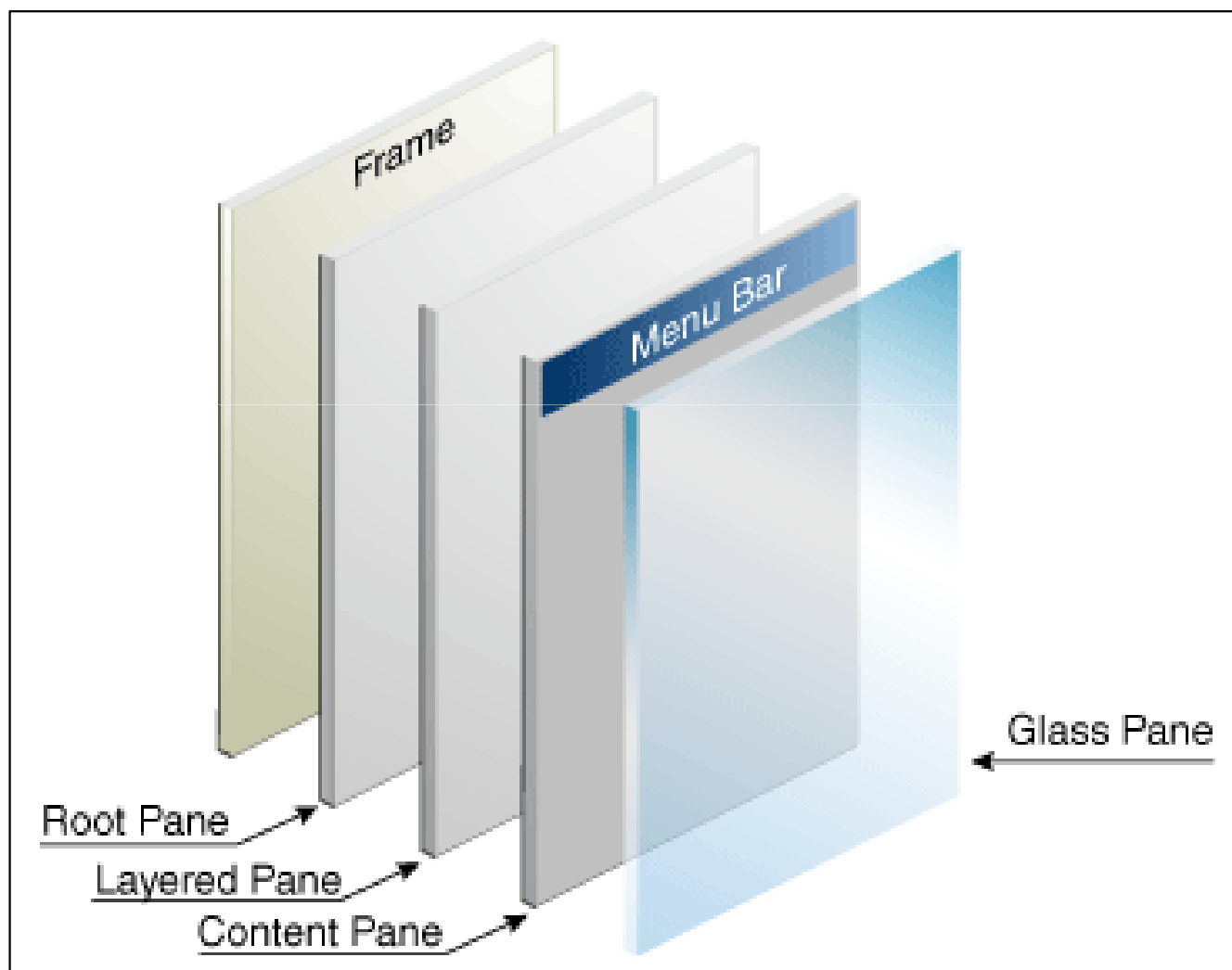
javax.swing.text.rtf

javax.swing.tree

javax.swing.undo



Níveis de Painéis



**KEEP
CALM
AND
PROGRAM
JAVA**

Apostila páginas 15 a 19

DÚVIDAS

A fatal exception 0E has occurred at 0028:C0011E36 in UXD UMM(01) + 00010E36. The current application will be terminated.

- * Press any key to terminate the current application.
- * Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _



Obrigado

fabio.aiub@gmail.com
about.me/fabiosperotto