

Enums

- Tipo de dados especial que permite variáveis com um conjunto fixo de constantes como valores possíveis

```
public enum DiaDaSemana {  
    SEGUNDA, TERÇA, QUARTA,  
    QUINTA, SEXTA, SÁBADO, DOMINGO  
}
```

```
...  
DiaDaSemana d = DiaDaSemana.SEGUNDA;  
...
```

Enums

```
...  
DiaDaSemana d;  
...  
if (d == DiaDaSemana.DOMINGO)  
    ...  
else  
    ...
```

```
...  
DiadaSemana d;  
...  
switch (d) {  
    case DOMINGO: ...  
    case SEGUNDA: ...  
    ...  
}
```

Nos **switch** os valores dos **case** não são qualificados com o nome do **Enum**.

Enums

- Enums são classes pelo que podem ter variáveis de instância e métodos.
- Cada constante terá os atributos e métodos.

```
public enum Elemento {  
    ALUMINIO, ANTIMONIO, ARGON, ARSENICO, AZUFRE;  
  
    private double pesoAtomico;  
  
    private double getPesoAtomico() {  
        return pesoAtomico;  
    }  
}
```

Necessário
';' !

Mas...
(está incompleto!)
como definir o peso de cada
elemento (como inicializar as
variáveis de instância)?!

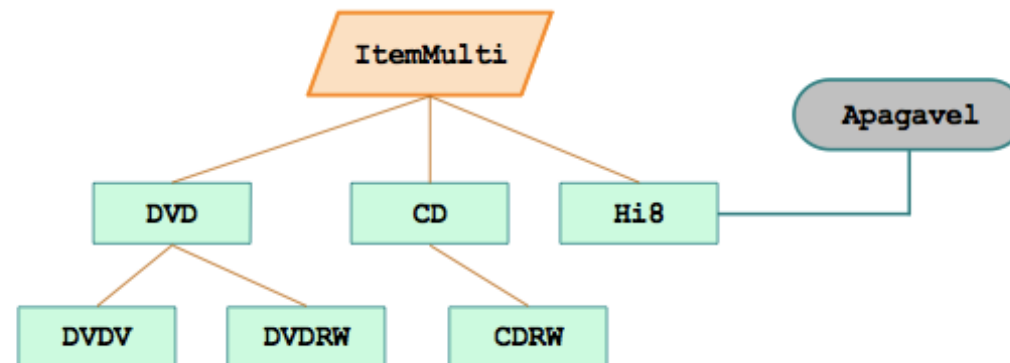
Enums

- Podemos inicializar as variáveis de instância na construção das constantes.
- Neste caso é necessário definir construtores (tem que ser **package** ou **private**)

```
public enum Elemento {  
    ALUMINIO(26.98), ANTIMONIO(121.8), ARGON(39.95),  
    ARSENICO(74.92), AZUFRE(32.06);  
  
    private double pesoAtomico;  
    Elemento (double p) {  
        this.pesoAtomico = p;  
    }  
  
    public double getPesoAtomico() {  
        return pesoAtomico;  
    }  
}
```

Ainda sobre interfaces...

- Uma hierarquia típica



```
public class Hi8 extends ItemMulti implements Apagavel {
    //
    private int minutos;
    private double ocupacao;
    private int gravacoes;
    . . . .
    // implementação de Apagavel
    public void apaga() { ocupacao = 0.0; gravacoes = 0; }
}
```

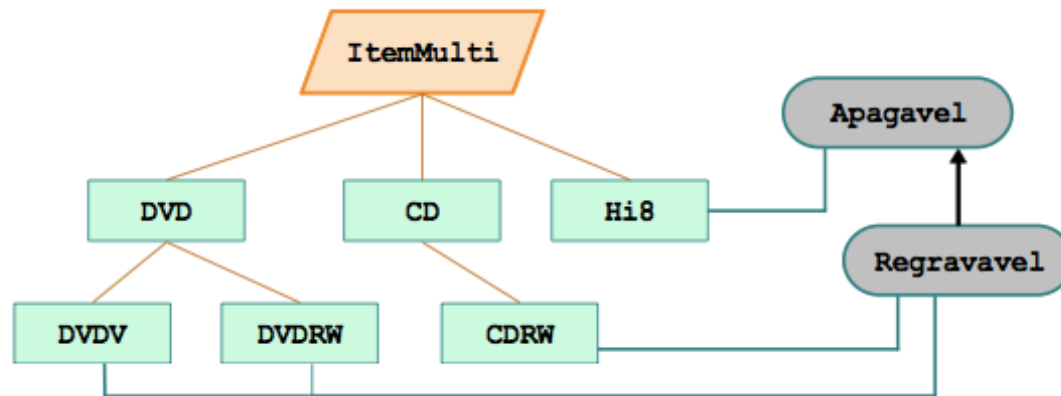
```
public interface Apagavel {
    /**
     * Apagar
     */
    public void apaga();
}
```

- Qualquer instância de Hi8 é também do tipo Apagavel, ou seja:

```
Hi8 filme1 = new Hi8("A1", "2005", "obs1", 180, 40.0, 3);  
Apagavel apg1 = filme1;  
apg1.apaga();
```

- no entanto, a uma instância de Hi8 que vemos como sendo um Apagavel, apenas lhe poderemos enviar métodos definidos nessa interface (i.e. nesse tipo de dados)

- Temos também a possibilidade de ter vários tipos de dados válidos para diferentes objectos.



- Por vezes, o que provavelmente acontece com **Regravavel**, a interface é apenas um marcador.

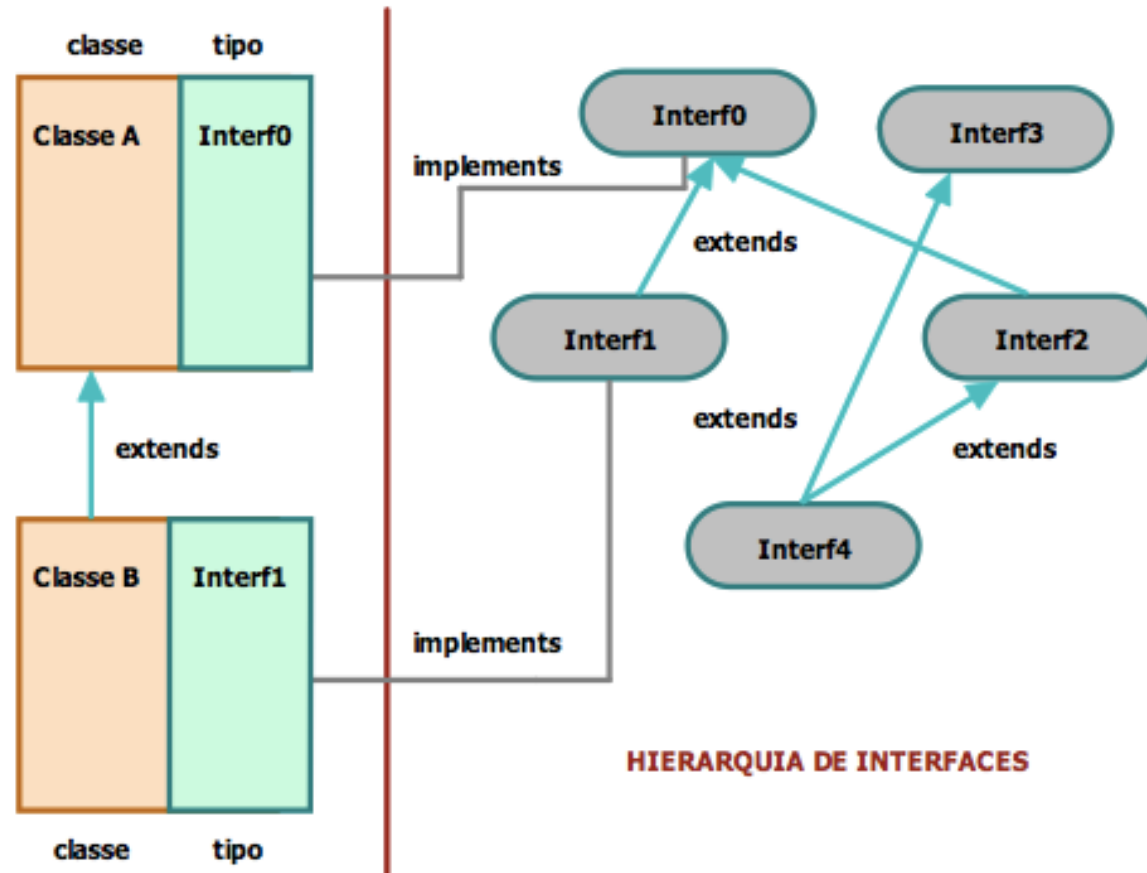
- A verificação de tipo pode ser feita da mesma forma que fazemos para as classes, com instanceof

```
ItemMulti[] filmes = new ItemMulti[ 500] ;  
// código para inserção de filmes no array....  
int contaReg = 0;  
for(ItemMulti filme : filmes)  
    if (filme instanceof Regravavel) contaReg++  
out.printf("Existem %d items regraváveis.", contaReg);
```

- na expressão acima não se está a validar a classe, mas sim o tipo de dados estático

- Do ponto de vista da concepção de arquitecturas de objectos, as interfaces são importantes para:
 - reunirem similaridades comportamentais, entre classes não relacionadas hierarquicamente
 - definirem novos tipos de dados
 - conterem a API comum a vários objectos, sem indicarem a classe dos mesmos (como no caso do JCF)

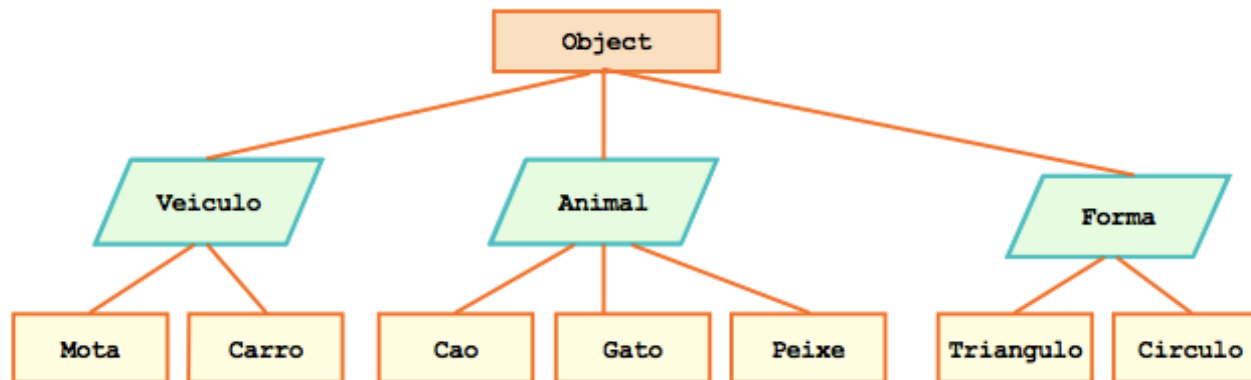
- O modelo geral é assim:



- onde coexistem as noções de classe e interface, bem assim como as duas hierarquias

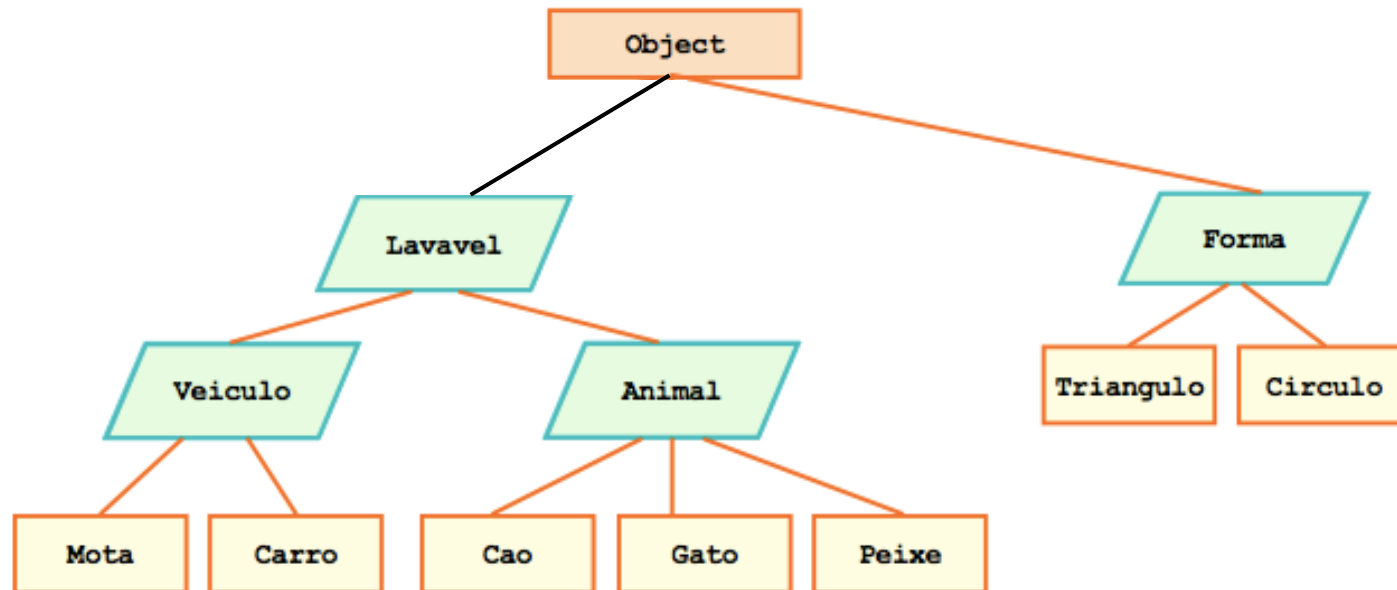
- Uma classe passa a ter duas vistas (ou classificações) possíveis:
- é subclasse, por se enquadrar na hierarquia normal de classes, tendo um mecanismo de herança simples de estado e comportamento
- é subtipo, por se enquadrar numa hierarquia múltipla de definições de comportamento abstracto (puramente sintático)

- Existem situações que apenas são possíveis de satisfazer considerando as duas hierarquias.



- se fosse importante saber os objectos desta hierarquia que poderiam ser lavados, então...

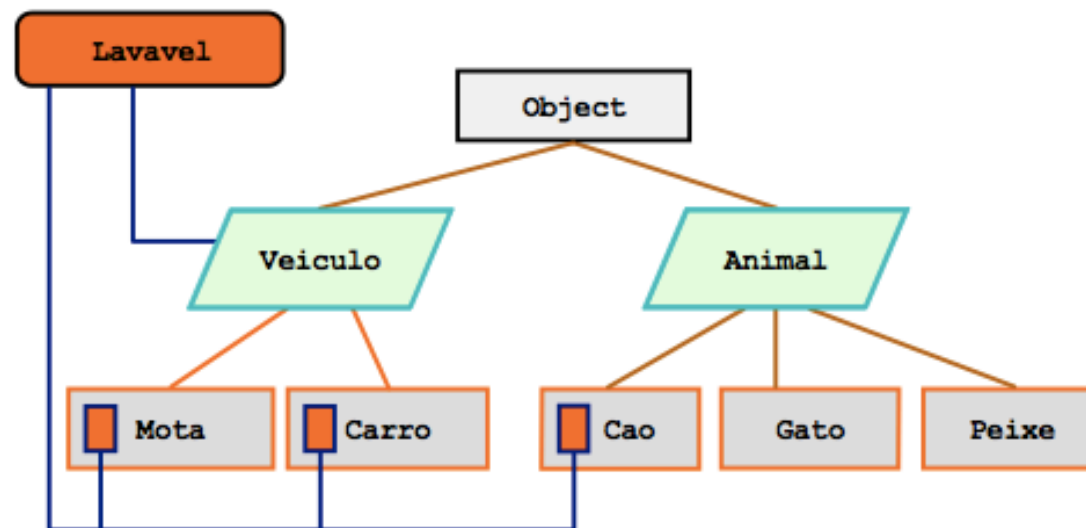
- Podíamos pensar em...



- no entanto, esta solução obrigaria objectos não “laváveis”, a ter um método `aLavar ()`

- Com a utilização de ambas as hierarquias poderemos ter:

```
public interface Lavavel {  
    public void aLavar();  
}
```



finalmente...

- Classes podem implementar multiplas interfaces
- Em Java8 as interfaces podem:
 - incluir métodos **static**
 - fornecer implementações por omissão dos métodos (*keyword* **default**)
- Functional Interface (Java8)
 - uma interface com um único método abstracto (e qualquer número de métodos **default**)
 - Instâncias criadas com expressões lambda e com referências a métodos ou construtores

Em resumo...

- As interfaces Java são especificações de tipos de dados. Especificam o conjunto de operações a que respondem objectos desse tipo
- Uma instância de uma classe é imediatamente compatível com:
 - o tipo da classe
 - o tipo da interface (se estiver definido)

Tratamento de Erros

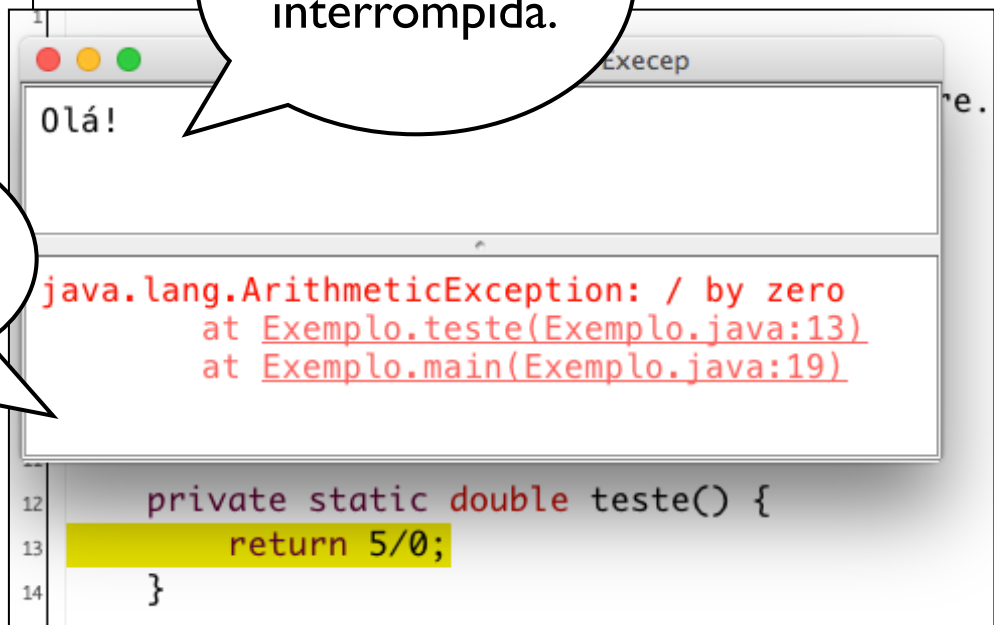
- Java usa a noção de *exceções* para realizar tratamento de erros
- Uma exceção é um *evento* que ocorre durante a execução do programa e que interrompe o fluxo normal de processamento

Exceções

```
10 public class Exemplo {  
11  
12     private static double teste() {  
13         return 5/0;  
14     }  
15  
16  
17     public static void main(String[] args) {  
18         System.out.println("Olá!");  
19         System.out.println(teste());  
20         System.out.println("Até logo!");  
21  
22     }  
23 }
```

O erro é propagado para trás pela stack de invocações de métodos.

A execução é interrompida.



```
12     private static double teste() {  
13         return 5/0;  
14     }
```

try catch

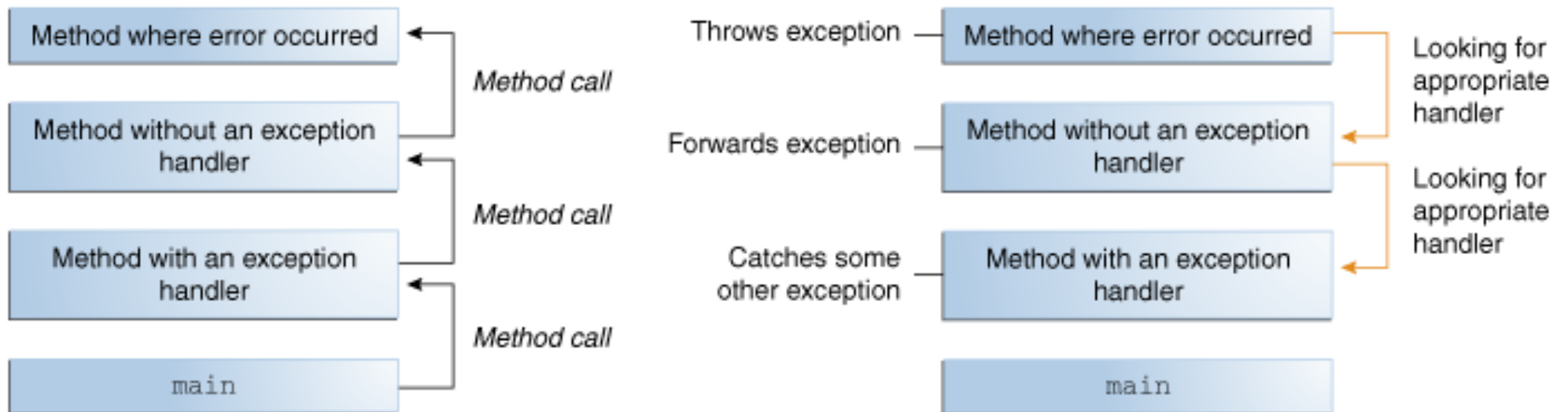
```
10 public class Exemplo {  
11  
12     private static double teste() {  
13         return 5/0;  
14     }  
15  
16  
17     public static void main(String[] args) {  
18         System.out.println("Olá!");  
19         try {  
20             System.out.println(teste());  
21         }  
22         catch (ArithmeticException e) {  
23             System.out.println("Ops! "+e.getMessage());  
24         }  
25         System.out.println("Até logo!");  
26     }  
27 }
```

A execução
retoma no **catch**.

BlueJ: Terminal Window - Excecep

```
Olá!  
Ops! / by zero  
Até logo!
```

Exceções



Criar Exceções

```
public class AlunoException extends Exception {  
    public AlunoException(String msg) {  
        super(msg);  
    }  
}
```

```
public static void main(String[] args) {  
    Opcoes op;  
    Aluno a;  
    int num;  
    do {
```

```
        op = lerOpcao();
```

```
        switch (op) {
```

```
            CONSULTAR:
```

```
                num = leNumero();
```

```
                try {
```

```
                    a = turma.getAluno(num);
```

```
                    out.println(a.toString());
```

```
                }
```

```
                catch (AlunoException e) {
```

```
                    out.println("Ops " + e.getMessage());
```

```
                }
```

```
                break;
```

```
            INSERIR:
```

```
                ...
```

```
        }
```

```
    } while (op != Opcoes.SAIR);
```

```
}
```

```
/**  
 * Obter o aluno da turma com número num.  
 *  
 * @param num o número do aluno pretendido  
 * @return uma cópia do aluno na posição  
 * @throws AlunoException  
 */  
public Aluno getAluno(int num) throws AlunoException {  
    Aluno a = alunos.get(num);  
    if (a == null)  
        throw new AlunoException("Aluno " + num + " não existe");  
    return a.clone();  
}
```

Obrigatório
declarar que lança
exceção.

Lança uma
exceção.

Vai tentar um
getAluno...

Apanha e
trata a
exceção.

Tipos de Exceções

- Exceções de *runtime*
 - Condições excepcionais interna à aplicação - ou seja, bugs!!
 - **RuntimeException** e suas subclasses
 - Exemplo; **NullPointerException**
- Erros
 - Condições excepcionais externas à aplicação
 - **Error** e suas subclasses
 - Exemplo: **IOError**
- Checked Exceptions
 - Condições excepcionais que aplicações bem escritas deverão tratar
 - Obrigadas ao requisito *Catch or Specify*
 - Exemplo: **FileNotFoundException**

Vantagens do uso de Exceções

- Separam código de tratamento de erros o código *regular*
- Propagação dos erros pela stack the invocações de métodos
- Junção e diferenciação de tipos de erros

Exemplo

Leitura/Escreita em ficheiros

EmpresaPOO

```
public void gravaObj(String fich) throws IOException {
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fich));
    oos.writeObject(this);
    oos.flush();
    oos.close();
}

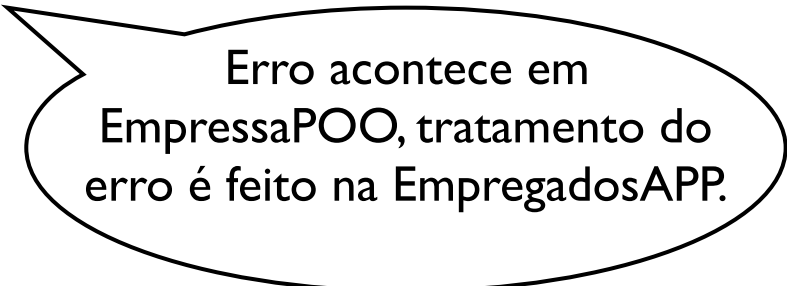
public static EmpresaPOO leObj(String fich) throws IOException, ClassNotFoundException {
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fich));

    EmpresaPOO te= (EmpresaPOO) ois.readObject();
    ois.close();
    return te;
}

public void log(String f, boolean ap) throws IOException {
    FileWriter fw = new FileWriter(f, ap);
    fw.write("\n----- LOG - LOG - LOG - LOG - LOG ----- \n");
    fw.write(this.toString());
    fw.write("\n----- LOG - LOG - LOG - LOG - LOG ----- \n");
    fw.flush();
    fw.close();
}
```


EmpregadosApp

```
private static void carregarDados() {  
    try {  
        tab = EmpresaP00.leObj("estado.tabemp");  
    }  
    catch (IOException e) {  
        tab = new EmpresaP00();  
        System.out.println("Não consegui ler os dados!\nErro de leitura.");  
    }  
    catch (ClassNotFoundException e) {  
        tab = new EmpresaP00();  
        System.out.println("Não consegui ler os dados!\nFicheiro com formato desconhecido.");  
    }  
    catch (ClassCastException e) {  
        tab = new EmpresaP00();  
        System.out.println("Não consegui ler os dados!\nErro de formato.");  
    }  
}
```



Erro acontece em
EmpresaPOO, tratamento do
erro é feito na EmpregadosAPP.

```

public static void main(String[] args) {
    carregarMenus();
    carregarDados();
    do {
        menumain.executa();
        switch (menu.getOpcao()) {
            case 1: inserirEmp();
                    break;
            case 2: consultarEmp();
                    break;
            case 3: totalSalarios();
                    break;
            case 4: totalGestores();
                    break;
            case 5: totalPorTipo();
                    break;
            case 6: totalKms();
                    break;
        }
    } while (menu.getOpcao() != 0);
    try {
        tab.gravaObj("estado.tabemp");
        tab.log("log.txt", true);
    }
    catch (IOException e) {
        System.out.println("Não consegui gravar os dados!");
    }
    System.out.println("Até breve!...");
}

```

Carregar dados no início
(erros são tratados dentro de
carregarDados).

Gravar dados
(e log) no fim (erros são
tratados aqui).