

Polimorfismo

Classes Abstratas

Classe Final

Tratamento de exceção

Jose.wellington@uniceub.br

Calendário

◀ agosto de 2013 ▶

D	S	T	Q	Q	S	S
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
-	-	-	-	-	-	-

◀ setembro de 2013 ▶

D	S	T	Q	Q	S	S
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

◀ outubro de 2013 ▶

D	S	T	Q	Q	S	S
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2

◀ novembro de 2013 ▶

D	S	T	Q	Q	S	S
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

◀ dezembro de 2013 ▶

D	S	T	Q	Q	S	S
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Agenda

- **Polimorfismo**
 - **Classe Abstrata**
 - **Classe Final**
- **Exceptions Try Cath**
- **Exercício**

Polimorfismo

Polimorfismo

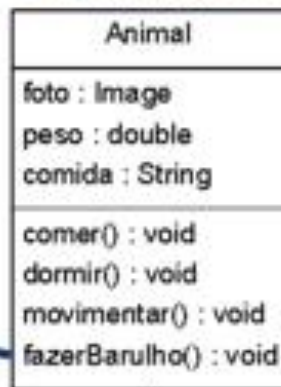
Polimorfismo

Comportamento Diferente

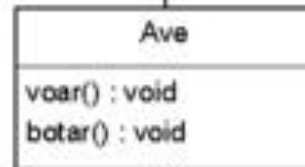
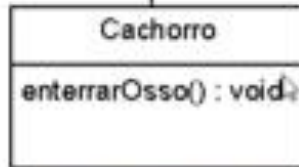
Au, Au !



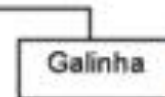
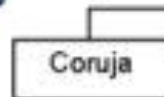
Superclasse



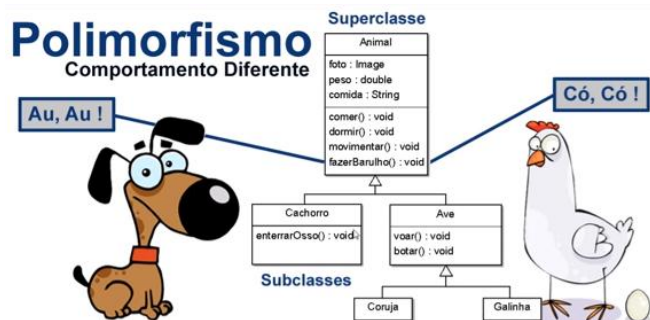
Có, Có !



Subclasses

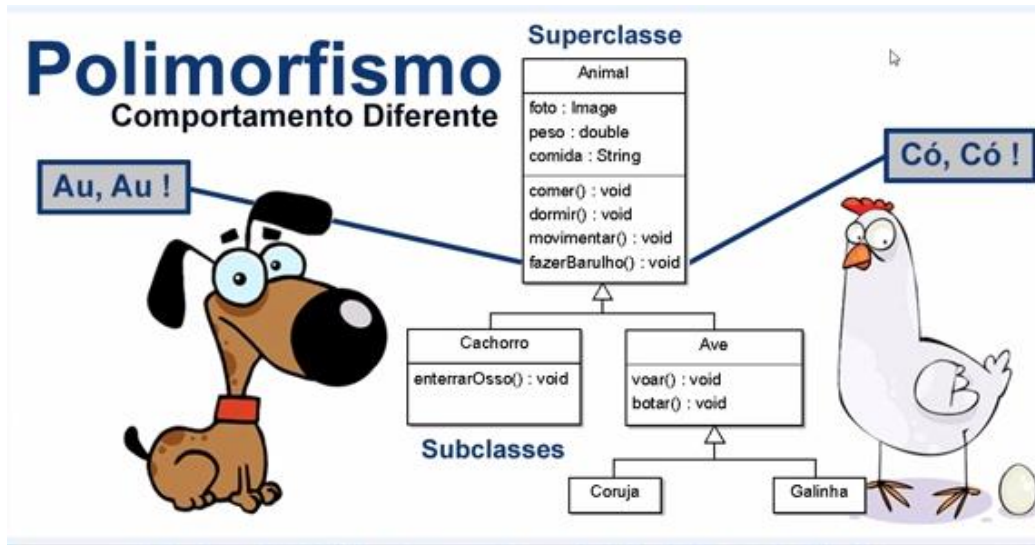


Polimorfismo



- ✓ **Polimorfismo**, que vem do grego "*muitas formas*".
- ✓ É o termo definido em linguagens orientadas a objeto - como o Java - para a possibilidade de se usar o mesmo elemento de forma diferente.
- ✓ Especificamente em Java, polimorfismo se encontra no fato de podemos modificar totalmente o código de um método herdado de uma classe diferente, ou seja, sobrescrevemos o método da classe pai.
- ✓ Portanto, polimorfismo está intimamente ligado a herança de classes.

Polimorfismo

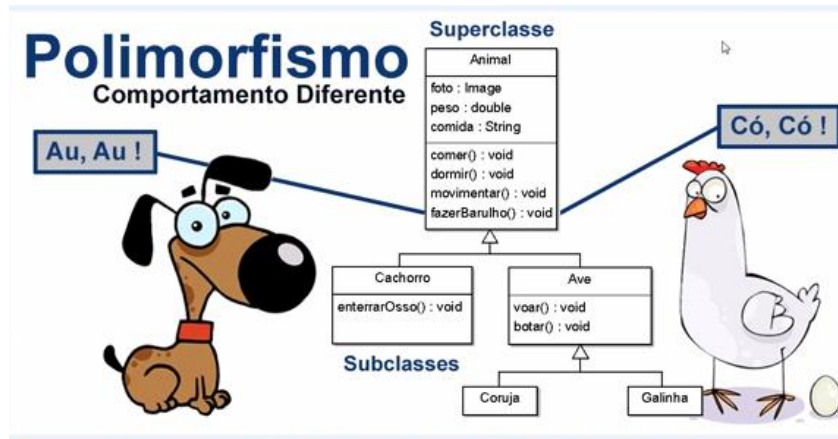


```
public class Animal {
```

```
    double peso;  
    String comida;
```

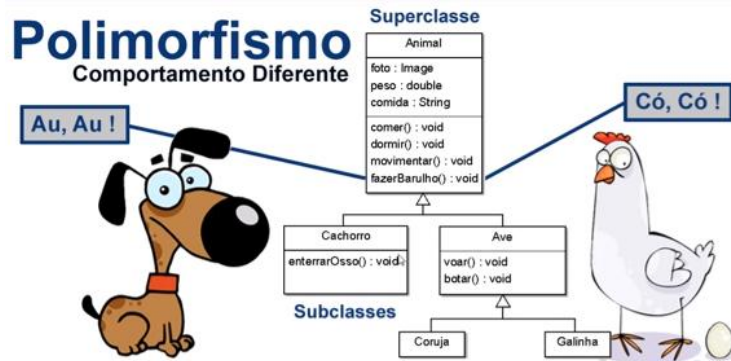
```
    void dormir(){System.out.println("Dormiu");}  
    void fazerBarulho(){System.out.println("Fazer Barulho")  
}
```

Polimorfismo



```
public class Animal {  
  
    double peso;  
    String comida;  
  
    public Animal(double peso, String comida) {  
        this.peso = peso;  
        this.comida = comida;  
    }  
  
    void dormir(){System.out.println("Dormiu");}  
    void fazerBarulho(){System.out.println("Fazer Barulho")  
}
```


Polimorfismo



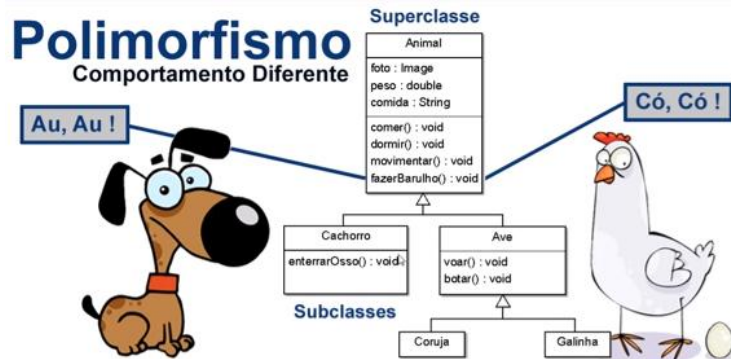
```
public class Cachorro extends Animal {
```

```
    public Cachorro() {  
        super(30, "Carne");  
    }
```



```
        void fazerBarulho() {  
            System.out.println("Au, Au !");  
        }  
    }
```

Polimorfismo



```

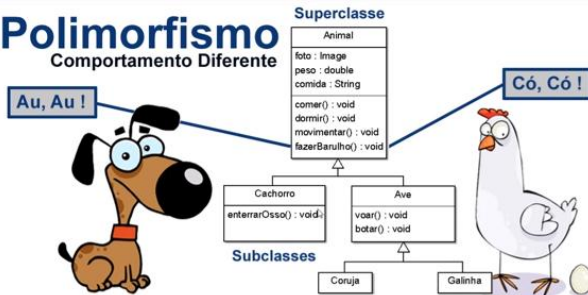
public class Galinha extends Animal {

    public Galinha() {
        super(2, "Milho");
    }

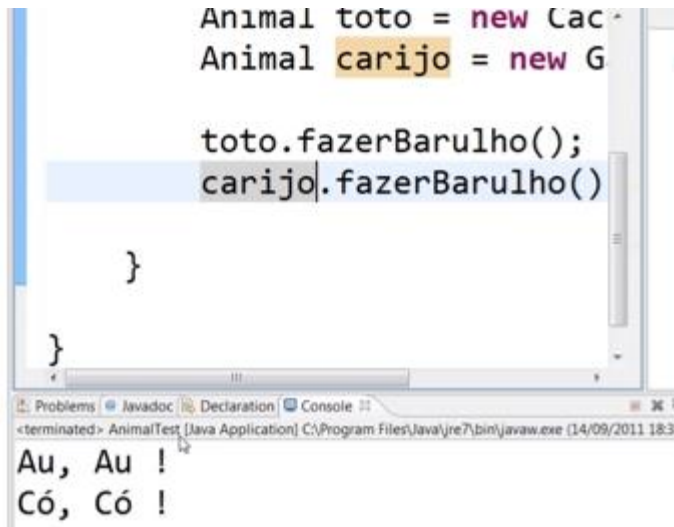
    void fazerBarulho() {
        System.out.println("Có, Có !");
    }
}
  
```

Polimorfismo

Polimorfismo Comportamento Diferente

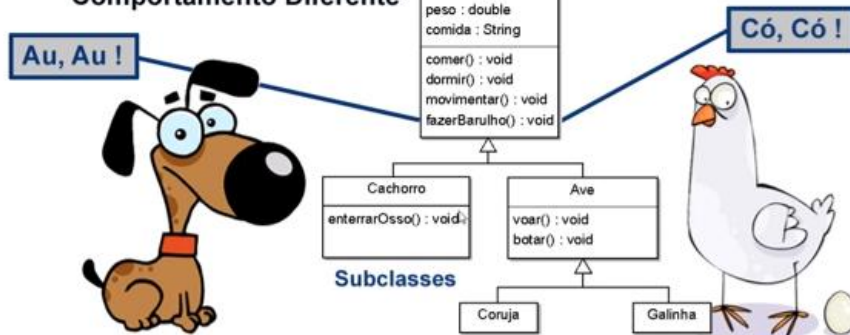


```
public class AnimalTest {  
  
    public static void main(String[] args) {  
  
        Animal toto = new Cachorro();  
        Animal carijo = new Galinha();  
  
        toto.fazerBarulho();  
        carijo.fazerBarulho()  
    }  
}
```



Outra Forma (Mais complexa) Sem Polimorfismo

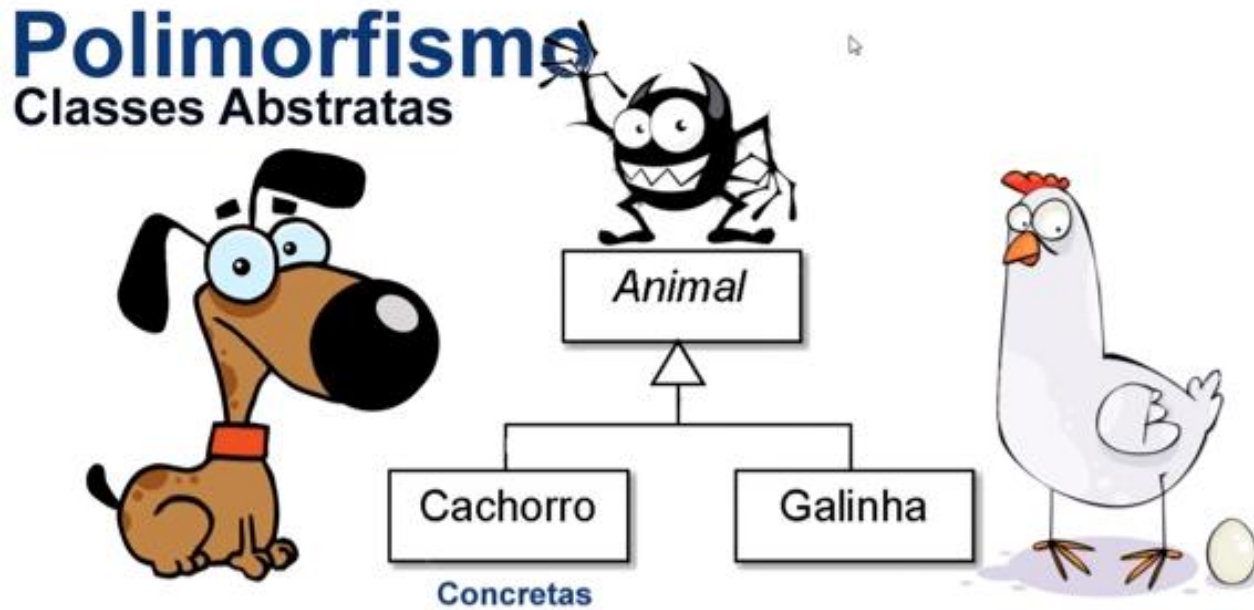
Polimorfismo Comportamento Diferente



```
public static void barulho(String animal) {  
    if(animal.equals("Cachorro")) {  
        System.out.println("Au, Au !");  
    } else if (animal.equals("Galinha")) {  
        System.out.println("Có, Có !");  
    }  
}
```

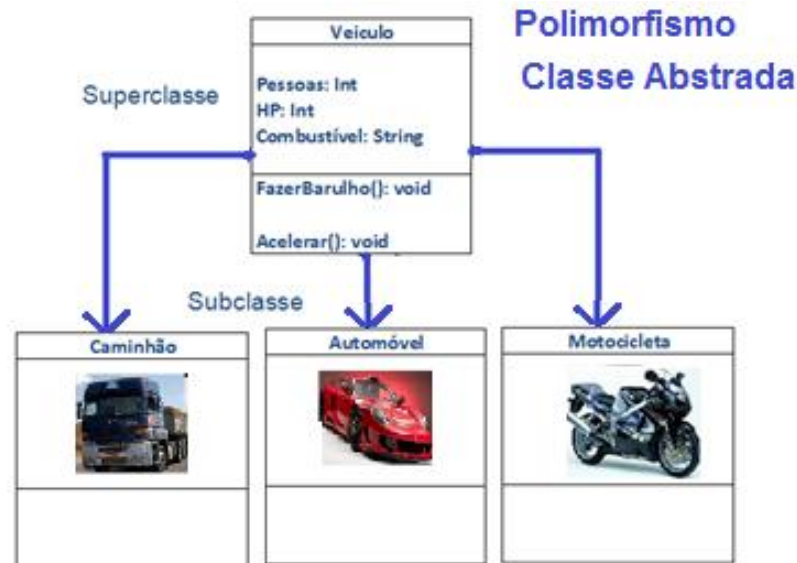
Classe Abstrata

Classe abstratas



- ❑ Usamos classe abstrata com o extends, ou seja é uma herança, e isso faz com que a subclasse herde o que for necessário de acordo com o escopo e visibilidade.

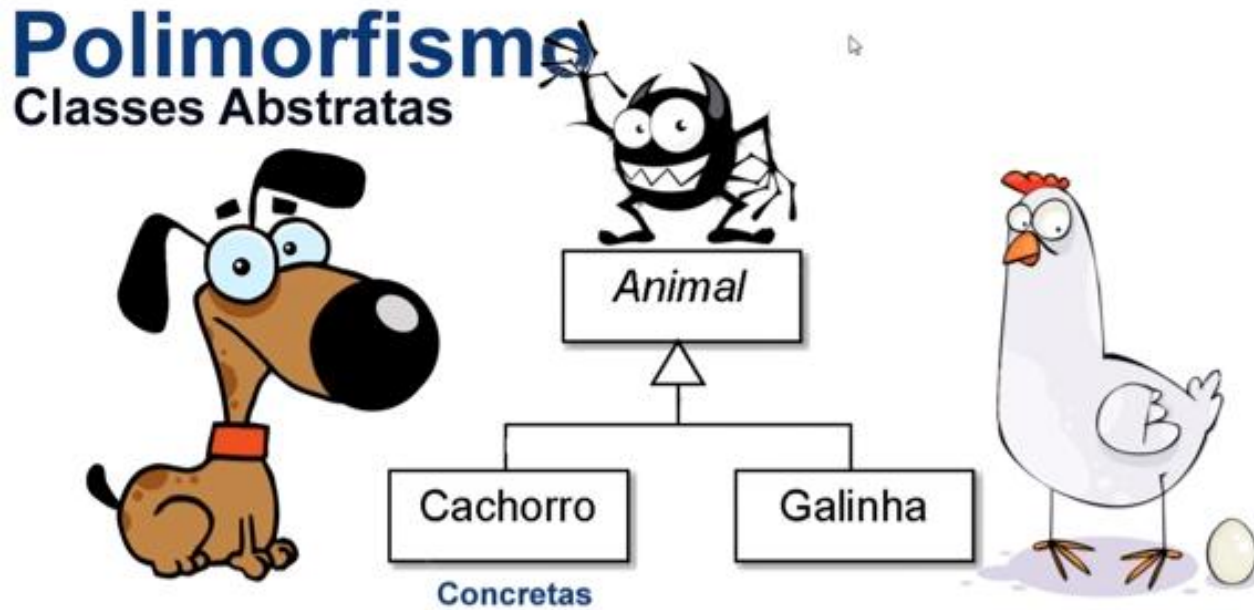
Classe abstratas



Imagine a classe **Veiculo** e suas subclasses **Automóvel**, **Moto** e **Caminhão**.

Faria mais sentido ter instâncias de **Automóvel**, **Moto** e **Caminhão**. , do que ter instâncias de **Veiculo**, que é uma abstração desses subtipos.

Classe abstratas



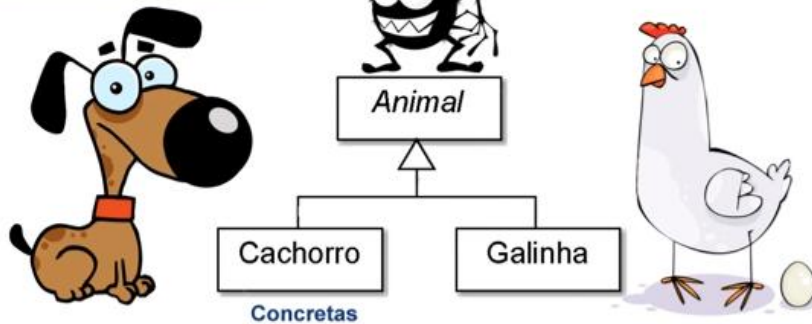
- ☐ **Classes Abstratas tem comportamento indefinido para ser implementado na criação de Objetos concretos.**
- ☐ **Não pode criar objetos através de classes abstratas**

Classe abstratas

Método abstrato

Polimorfismo

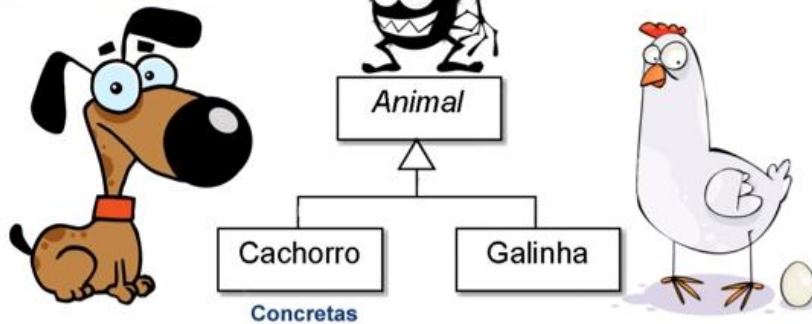
Classes Abstratas



```
public abstract class Animal {  
  
    double peso;  
    String comida;  
  
    void dormir() {  
        System.out.println("Dormiu");  
    }  
  
    abstract void fazerBarulho();  
}
```

Classe abstratas

Polimorfismo Classes Abstratas



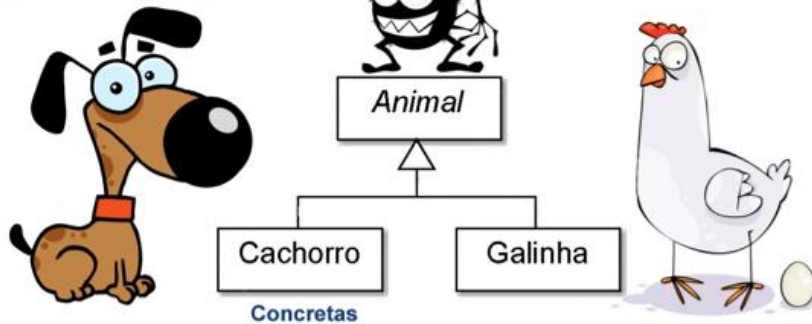
- ❑ A classe **ANIMAL** não pode ser instanciada

```
public class AnimalTest {  
  
    public static void main(String[] args) {  
        //Animal animal = new Animal();  
        Animal cachorro = new Cachorro();  
        Animal galinha = new Galinha();  
    }  
}
```

Classe abstratas

Polimorfismo

Classes Abstratas



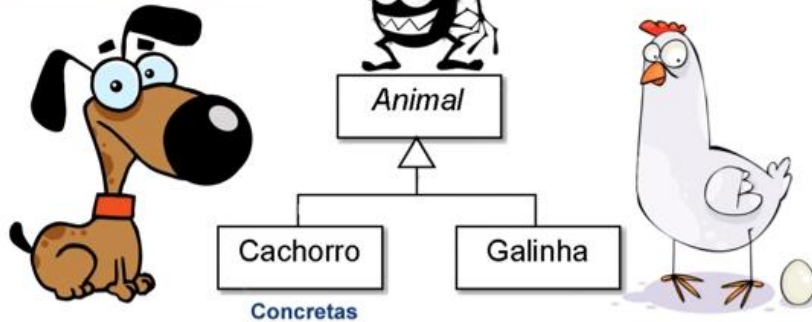
- ❑ A classe **ANIMAL** **não** pode ser instanciada
- ❑ Só pode Extends

```
public class Cachorro extends Animal {  
  
    void fazerBarulho() {  
        System.out.println("Au, Au !");  
    }  
}
```

Classe abstratas

Polimorfismo

Classes Abstratas



- ❑ Objetos só pode ser implementadas por Classe Concreta.

```
public class AnimalTest {  
  
    public static void main(String[] args) {  
        //Animal animal = new Animal();  
        Animal cachorro = new Cachorro();  
        Animal galinha = new Galinha();  
    }  
}
```

Classe Final

Classe Final



```
public final class Cachorro extends Animal {  
    void fazerBarulho() {  
        System.out.println("Au, Au !");  
    }  
}
```

- ❑ Uma vez aplicada a classe não poderá ser herdada

Classe Final

```
public abstract class Animal {  
  
    double peso;  
    String comida;  
    ↓  
    final void dormir() {  
        System.out.println("Dormiu");  
    }  
  
    abstract void fazerBarulho();  
}
```

- ❑ Método Final não poderá ser herdado

Exceptions Try Cath

Exceptions

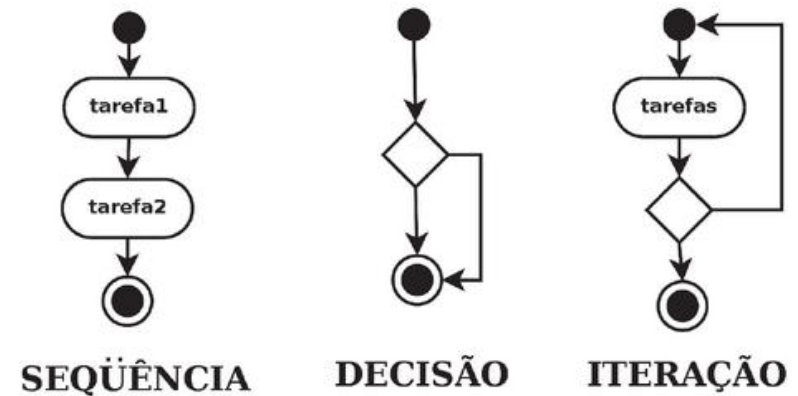
- Um velho **axioma** relacionado ao desenvolvimento de softwares diz que **80%** do trabalho (esforço necessário para identificar e manipular erros) são usados em **20%** do tempo de desenvolvimento.



**UN AXIOMA
ES UN PRINCIPIO
ES UNA VERDAD
QUE NO SE DISCUTE
Y NO NECESITA
DEMOSTRACION**

Exceptions

- Na **programação estruturada**, desenvolver programas que lidam com erros é monótono e **transforma o código-fonte do aplicativo em um emaranhado confuso.**



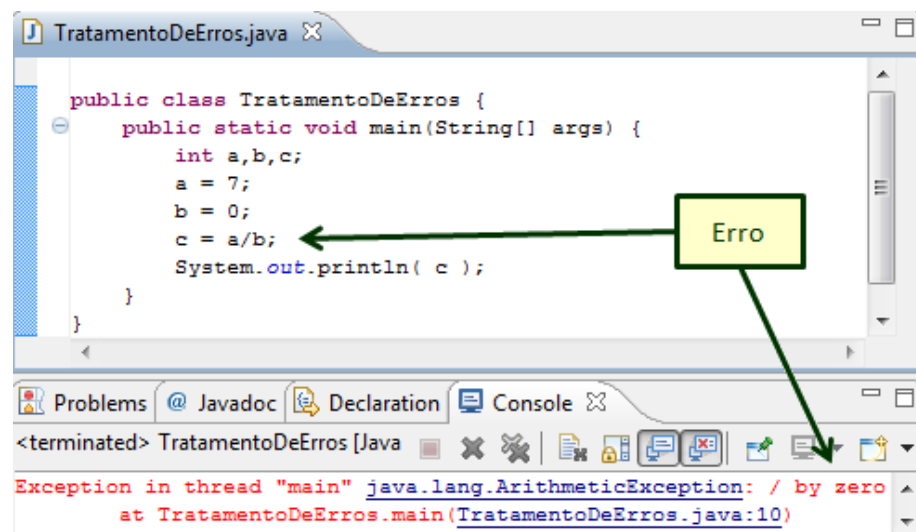
Exceptions

- Na **programação orientada a objetos**, aqui especificamente na Linguagem **Java**, existe um mecanismo sofisticado para manipulação de erros que produz códigos de manipulação eficientes e organizados: a ***manipulação de exceções***.



Vantagens da manipulação de exceções

- ❑ **Fácil detecção de erros** sem a escrita de um código especial para testar valores retornados;
- ❑ Permite manter um código de manipulação de exceções **nitidamente separado do código** que gerará a exceção
- ❑ Permite que o mesmo código de manipulação de exceções lide com as diferentes exceções possíveis.



```
TratamentoDeErros.java X

public class TratamentoDeErros {
    public static void main(String[] args) {
        int a,b,c;
        a = 7;
        b = 0;
        c = a/b;
        System.out.println( c );
    }
}

Problems @ Javadoc Declaration Console X
<terminated> TratamentoDeErros [Java]
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at TratamentoDeErros.main(TratamentoDeErros.java:10)
```

Tratamento de erro

```
try {  
    // executa até linha onde ocorrer exceção  
} catch (TipoExcecao ex) {  
    // executa somente se ocorrer TipoExcecao  
}  
finally {  
    // executa sempre ...  
}  
  
// executa se exceção for capturada ou se não ocorrer
```

Exemplo

```
import java.util.Scanner;

public class Excecao {

    public static void main(String[] args) {
        try {
            Scanner entrada = new Scanner(System.in);

            System.out.println("Entre com o 1 nr = ");
            int nr1 = entrada.nextInt();
            System.out.println("Entre com o 2 nr = ");
            int nr2 = entrada.nextInt();

            int resultado = nr1/nr2;

            System.out.println("Resultado = " + resultado);}

        catch (Exception e ) {
            System.out.println(e.getMessage());
        }

        finally {
            System.out.println(" volta para o menu ");
        }

    }
}
```

Exemplo

⋮ Saída - Excecao (run)



run:



Entre com o 1 nr =

10



Entre com o 2 nr =

0



/ by zero

volta para o menu

CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)

|

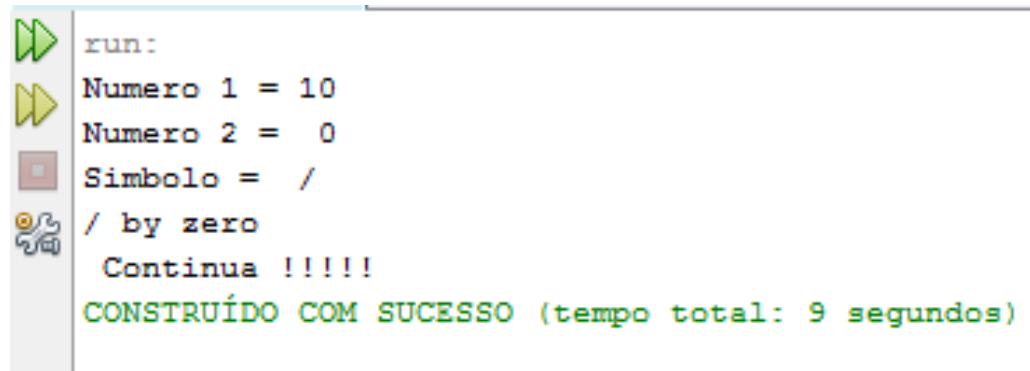
Exercício

Exercício 21

Faça um programa que entre com dois números e o símbolo de Operação (+, -, *, /) e mostre o resultado **USANDO O TRATAMENTO DE EXEÇÃO**

Sabendo que:

- 1) Entre com Número 1 e Número 2
- 2) Entre com o Símbolo de Operação (+,-,*,/)
- 3) Mostre o resultado



```
run:
Numero 1 = 10
Numero 2 = 0
Simbolo = /
/ by zero
Continua !!!!!
CONSTRUÍDO COM SUCESSO (tempo total: 9 segundos)
```

Agenda

- **Polimorfismo**
 - **Classe Abstrata**
 - **Classe Final**
- **Exceptions Try Cath**
- **Exercício**