
RUBY

Características

- Linguagem interpretada.
- Multiparadigma.
- Tipagem forte e dinâmica.
- Gerenciamento automático de memória.
- Multiplataforma.
- Open-source.

Métodos "attribute accessors"

- Ruby:

- class Bichos
  - attr_accessor :animais
- end
- exemplo = Bichos.new()
- exemplo.animais = ["cachorro","gato","cavalo"]
- print exemplo.animais

- Java:

- import java.util.*;

- public class Bichos{
- Vector animais = new Vector();

- public String getAnimais(){
- return this.animais.toString();
- }

  - public void setAnimais(String animal){
    - this.animais.add(animal);
  - }
- }

Métodos que retornam true ou false terminam com interrogação

- Ruby:
- animais = ["cachorro","gato","cavalo"]
- print animais.include? "gato"

Métodos que alteram o objeto são terminados com ponto de exclamação

- Ruby:
- animais = ["cachorro","gato","cavalo"]
- animais.slice!(1)
- print animais.include? "gato"

Métodos Slice! e Include? (java)

- Java:

```
import java.util.*;
public class Main{
   public static void main(String[] args) {
      List animais = new ArrayList();
      boolean achou = false;

      animais.add("cachorro");
      animais.add("gato");
      animais.add("cavalo");
 //====== slice! ======
      animais.remove(1);
//==================
//====== include? ======
      for(int i = 0; i<array.size(); i++){
        if(animais.get(i).equals("gato")){
          achou = true;
          break;
        }
      }
//====================
      System.out.println(achou);
   }
}
```

Variáveis como Instâncias

- Ruby:

```
def pares(limite)
    1.upto(limite) {
      |i| puts i if i % 2 == 0
    }
  end
```

- Java:

```
public void pares(int limite){
    for(int i = 1; i<= limite; i++){
      if(i % 2)System.out.println(i);
    }
 }
```