FUNDAMENTOS DA COMPUTAÇÃO

PROF. JOSENALDE OLIVEIRA

josenalde@eaj.ufrn.br

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Linguagens de programação: *conjuntos de palavras-chave, símbolos e um sistema de regras para construir declarações pelas quais os seres humanos podem comunicar instruções para o computador executar* – ***softwares básicos, firmwares e aplicações***

Sintaxe: conjunto de regras associado a uma linguagem de programação: estabelece como os símbolos são combinados em declarações capazes de levar instruções sensatas à CPU – normalmente associado à FORMA (palavras reservadas, comandos, recursos etc.)

```
class HelloWorld
{ static void Main() {
     System.Console.WriteLine("Hello, World!"); } }
```

C#

```
program HelloWorld(output);
begin WriteLn('Hello World!'); end
```

Object Pascal

```
int main() { cout >> "Hello World"; }
```

C++

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Análise sintática (léxica + sintática):

- **léxica:** ler código fonte caractere a caractere e identifica palavras (lexemas) e depois os classifica em símbolos, palavras reservadas, identificadores etc. (tokens) de acordo com o alfabeto/gramática da linguagem – remove comentários

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Exemplo: if (a < 100) x = 5;

[if] [(] [id, a] [<] [num,100] [)] [id, x] [=] [num,5] [;]: lista de tokens (já com (tipo) atribuído (identificador, numeral, reservado, ;, identação, ...). Passa lista ao analisador sintático (parser)

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Exemplo: if x == y then

```
        z = 1;
else
        z = 2;
```

[if] [x] [==] [y] [then][\n] [z] [=] [1] [;] [\n] [else] [\n] [z] [=] [2] [;] <EOF>

Lexemas acima são consultados na **Tabela de Símbolos** para ter tipo atribuído

Tipos: identificador: strings de letras ou dígitos, iniciados por letra

Numeral: string de dígitos

Espaço em branco: string de brancos, quebra de linha, tabs ou comentários

Palavras reservadas: while, if, do, int, float, double, else...

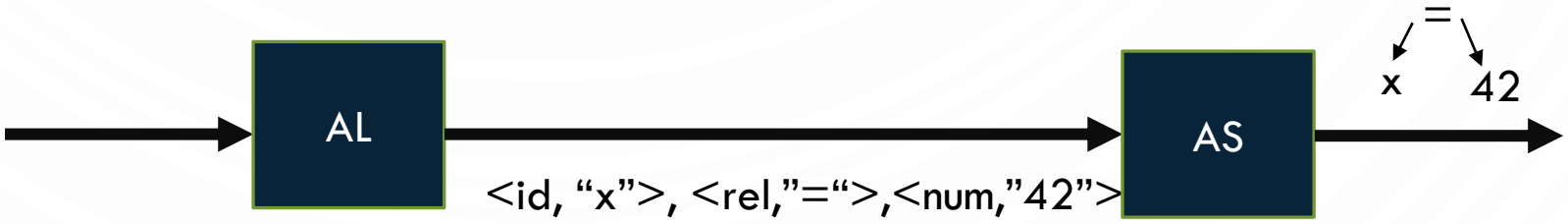
LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Para o código abaixo, classifique e contabilize os tokens (id, relacional, num, espaço, reservado, outros

```
x = 0;\nwhile (x < 10) {\n\tx++;\n}\n
```

O analisador léxico envia o token, par (substring): tipo,substring, para o analisador sintático checar se a ordem está coerente com a gramática

Exemplo: x=42



LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

As linguagens C e C++ usam conceito similar de token para separar substrings de uma string, de acordo com determinado símbolo: strtok, find

```
int main() {
    FILE *fp;
    char linha[100];
    int vetor[17];

    int L = 1, i = 0, j = 6;
    char *leitura;
    fp = fopen("numeros.txt", "r");
    while (!feof(fp)) {
        leitura = fgets(linha, 100, fp);
        if (leitura) {
            printf("Linha %d: %s", L, linha);
            char *token = strtok(linha, " ");
            while (token != NULL && i < j) {
                vetor[i++] = atoi(token);
                token = strtok(NULL, " ");
            }
            j += i;
        }
        L++;
    }
    i = 0;
    for (; i < 17; i++) printf("%d\t", vetor[i]);
    return 0;
}
```

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

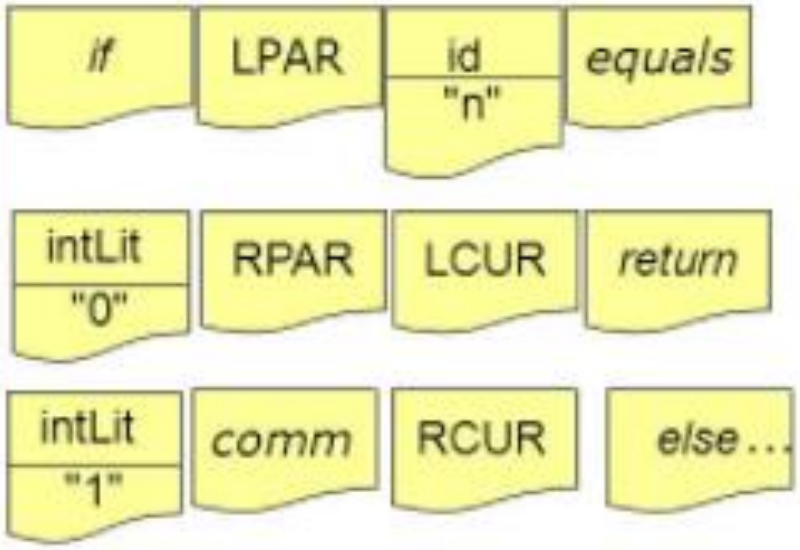
Na prática, os códigos são equivelentes. Mesmo o analisador léxico identificando os caracteres não imprimíveis ‘ ‘ (espaço), ‘\n’, ‘\t’, utiliza em alguns casos função para remover os espaços (trim). No caso de identificadores não é possível esta remoção.

```
int main(){
   int x=2,y; if(x>1){y=1;}
}
```

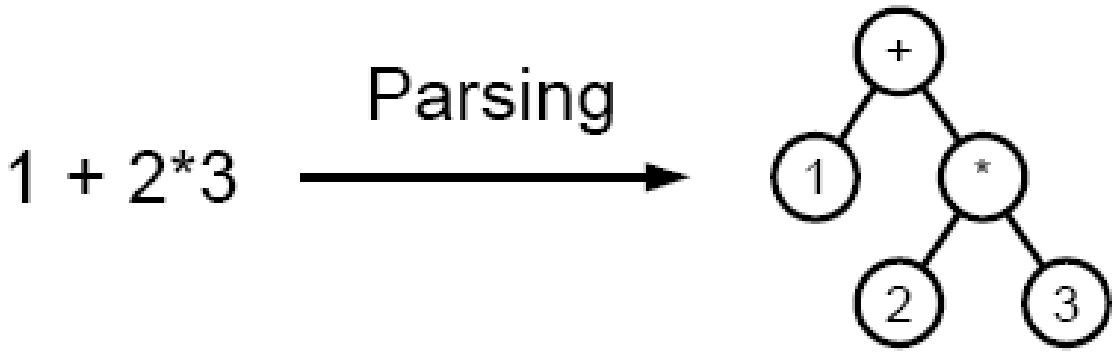
```
int main() {
   int x=2,y;
    if (x>1) {
       y=1;
    }
}
```

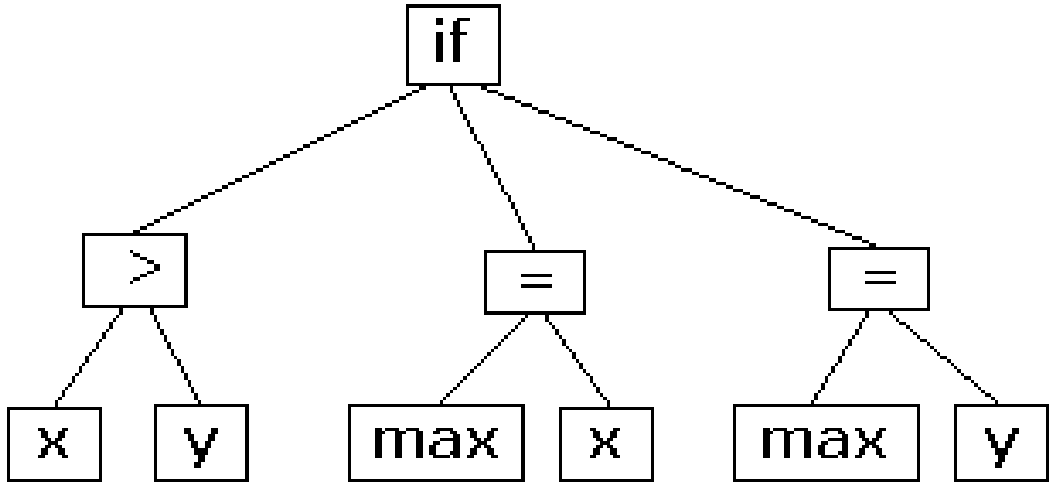
LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

```
if (n == 0) {
    return 1;
} else {
    ...
}
```



LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS
ANÁLISE SINTÁTICA





LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Semântica: de um modo geral é o estudo do significado das coisas (conteúdo)

Imagine um profissional que se expressa muito bem (oratória), porém não sabe do que fala – é como um sistema com um lindo front-end (interface, GUI), mas com muitos bugs em suas funcionalidades; da mesma forma, pode-se ter um artefato que funciona corretamente, porém com código mal organizado, não comentado, de difícil compreensão pelo analista, outros programadores ou mesmo o próprio autor (contrário: entende bem do assunto (conteúdo), mas não se expressa, comunica bem (forma))

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

Semântica/Sintaxe: exemplo: somar as vendas de uma quinzena, recebendo como parâmetros as vendas da semana anterior e da semana atual

```
public int M0998_Sma(int M0998_1, int M0998_2)
{
return M0998_1 + M0998_2;
}
```

Facilitar manutenção, revisão e teste
Atenção: Desenvolvimento em EQUIPE

```
public int somaResultadoQuinzena (int aValorSemanaAnterior, int avalorSemanaAtual)
{
int resultadoQuinzena = aValorSemanaAnterior + avalorSemanaAtual;
return resultadoQuinzena;
}
```

LINGUAGENS DE PROGRAMAÇÃO E PARADIGMAS

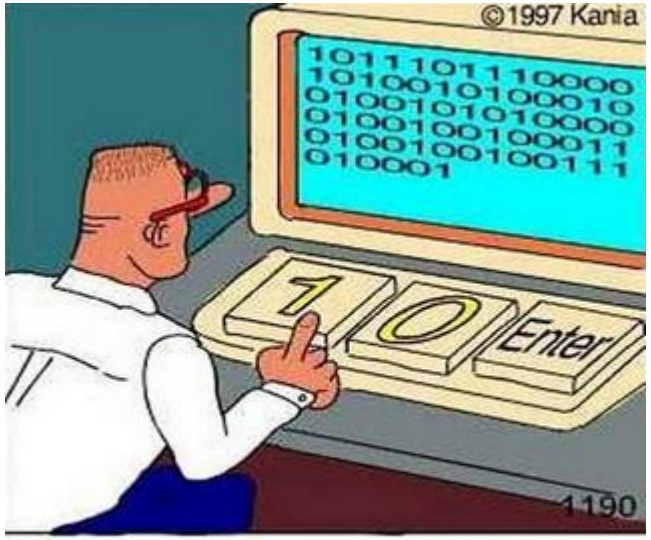
Erros semânticos também podem ser associados à verificação do sentido de expressões, mesmo com a sintática correta. Por exemplo:

while (x > 2).... (neste caso está correto, pois o comando while espera uma expressão lógica), mas while (x + 2) está incoerente semânticamente, mas mesmo assim o compilador não dá erro, pois basta que o argumento seja >=1 para considerar TRUE!

Alguns também consideram erros como tentativa de acessar arquivos que não existem como erro semântico, só percebido em tempo de execução

EVOLUÇÃO DAS LINGUAGENS

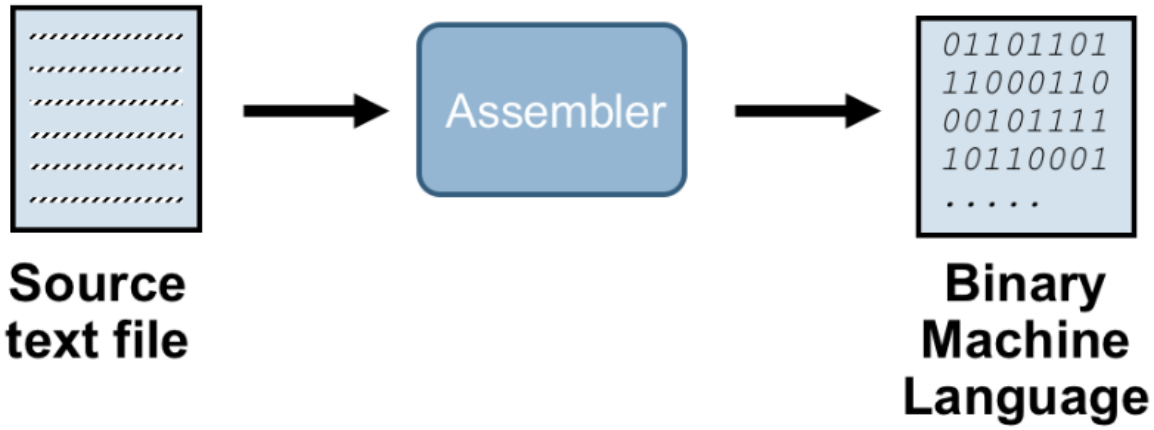
Primeira geração: chamada de linguagem de máquina, que exige o uso de símbolos binários (0 e 1). Essa é a linguagem da CPU, que manipula sinais elétricos nível ALTO (1, 5V TTL por exemplo) e nível BAIXO (0-0,8V TTL por exemplo). Os arquivos texto traduzidos para forma de conjuntos binários podem ser lidos por quase todas as plataformas de sistemas computacionais.



LOS VERDADEROS PROGRAMADORES PROGRAMAN EN BINARIO

EVOLUÇÃO DAS LINGUAGENS

Segunda geração: substituir dígitos binários por símbolos que os programadores poderiam entender com mais facilidade. Essas linguagens usam, por exemplo, um código como A para adicionar, MVC para mover e assim por diante. Também são conhecidas como linguagens de montagem (os programadores são chamados montadores) que as traduzem em código de máquina.



EVOLUÇÃO DAS LINGUAGENS

Terceira geração: usam declarações e comandos similares ao inglês, mais fácil de aprender (curva de aprendizagem) e usar do que linguagem de máquina e de montagem. Cada declaração é traduzida em diversas instruções de linguagem de montagem (compilador), e por fim, são executadas pela CPU. Basic, Cobol, C e Fortran são exemplos.

Quarta geração: enfatizam resultados em vez da forma de escrever. Podem ser usadas sem treinamento em linguagens de programação das gerações anteriores e são muito usadas para acessar informações em bases de dados, como PowerBuilder, Essbase, Forte, Focus, Powerhouse, SAS e SQL.

EVOLUÇÃO DAS LINGUAGENS

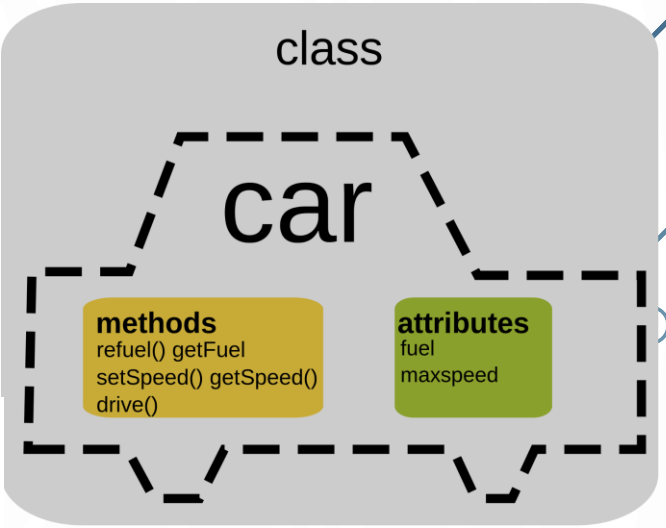
Quinta geração: usa interface de desenvolvimento visual ou gráfica (IDE), com compilador 3GL ou 4GL. Java, Visual Basic, PC COBOL, Visual C++ (Atualmente Visual Studio), .NET, C#, Javascript...

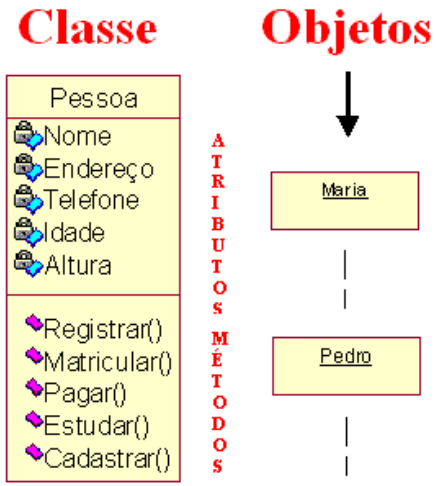
Linguagens Orientadas a Objeto (OO): objeto – agrega dados e ações que nele podem ser efetuadas. Exemplo: funcionário (dados pessoais etc., operações como folha de pagamento etc.)

Programa é construído em módulos – blocos com dados, instruções e procedimentos. Podem ser reutilizados. Mais populares C++, Java, Smalltalk, C#, Python; *no dia a dia as pessoas interagem com objetos, com comportamento próprio e características (atributos)*

ORIENTAÇÃO OBJETOS

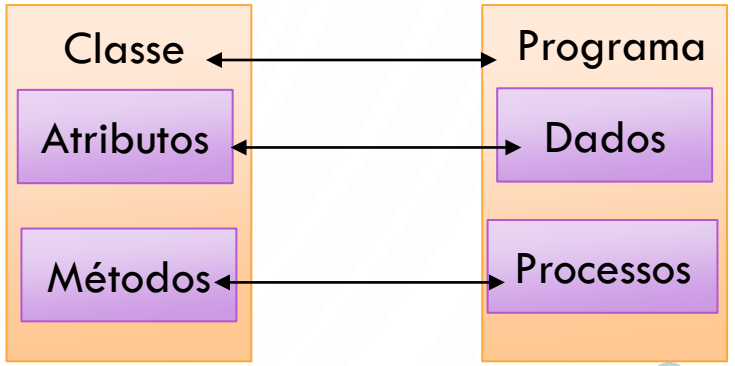
```cpp
#include <iostream>
#include <string>
using namespace std;

class Pessoa {
    private:
        string nome;
        unsigned short idade;
        float peso;
    public:
        void setIdade(unsigned short idade) {
            this->idade = idade;
        }
        unsigned short getIdade() {
            return idade;
        }
};

int main() {
    Pessoa a, b;
    a.setIdade(20);
    b.setIdade(18);
    cout << "Idade de a: " << a.getIdade() << endl;
    cout << "Idade de b: " << b.getIdade() << endl;
}
```





ORIENTAÇÃO OBJETOS X ESTRUTURADA

Vê o software como coleção de objetos que interagem entre si e apresentam características próprias, com processos (operações ou métodos) e dados (atributos)



- Paradigma tradicional (estruturado): foco nas funções que o sistema deve realizar; na modelagem descobre-se os processos, para depois descobrir quais dados são necessários e qual a relação entre eles

Abordagens **top-down** e **bottom-up**

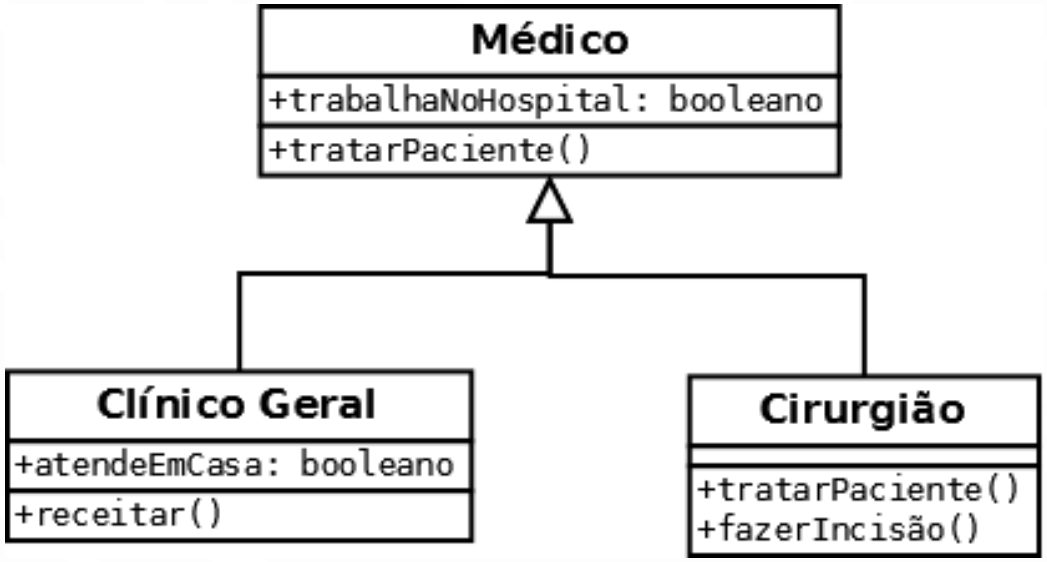
CARACTERÍSTICAS ORIENTAÇÃO OBJETOS

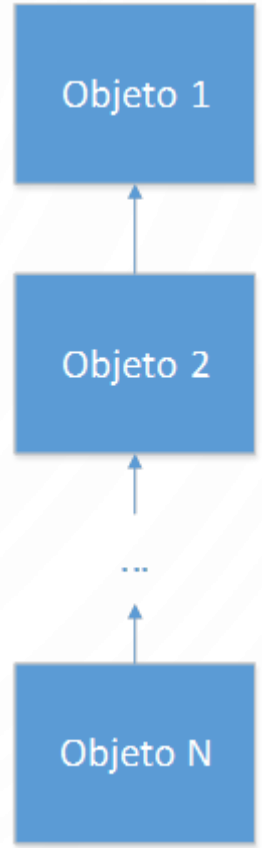
- **Abstração:** representar objeto real identificando sua identidade, propriedades (atributos) e métodos (operações sobre o objeto)

- **Encapsulamento:** espécie de caixa preta. Esconde a implementação da utilização de fato dos objetos. A leitura (get) e escrita (set) em propriedades dos objetos é feita pelos métodos getters e setters. Exemplo: ligar botão da TV (o que acontece por trás?)

CARACTERÍSTICAS ORIENTAÇÃO OBJETOS

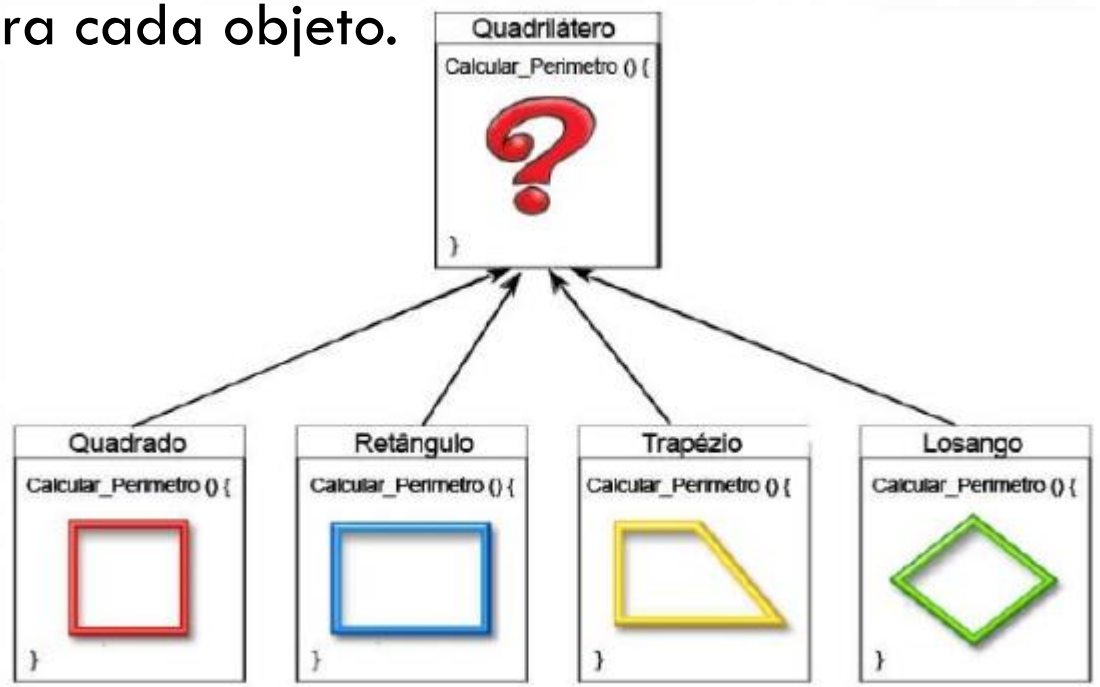
- Herança: herdar características de outras classes de objetos, reutilizar código





CARACTERÍSTICAS ORIENTAÇÃO OBJETOS

- Polimorfismo: alterar o funcionamento de um método herdado de um objeto pai. Exemplo: classe eletrodoméstico com a ação Ligar(). Suponha dois objetos TV e Geladeira, que não são ligados da mesma forma, portanto o método Ligar() precisa ser reescrito para cada objeto.



ORIENTAÇÃO A EVENTOS

Normalmente sistemas com interface gráfica: resposta à ações do usuário (teclado, mouse etc.) – onMove, onChange, onFocus, onClick etc.

C# UNITY (GAMES)

```
1 using UnityEngine;
2 using System.Collections;
3
4 [System.Serializable]
5 public class DataClass {
6     public int myInt;
7     public float myFloat;
8
9 }
10
11 public class DemoScript : MonoBehaviour {
12
13     public Light myLight;
14     public DataClass myClass;
15
16     void Awake () {
17         int myVar = AddTwo(9,2);
18         Debug.Log(myVar);
19
20     }
21
22     void Update () {
23         if (Input.GetKeyDown ("space")) {
24             MyFunction ();
25
26         }
27
28     }
29
30     void MyFunction () {
31         myLight.enabled = !myLight.enabled;
32     }
```

MIT APP INVENTOR2