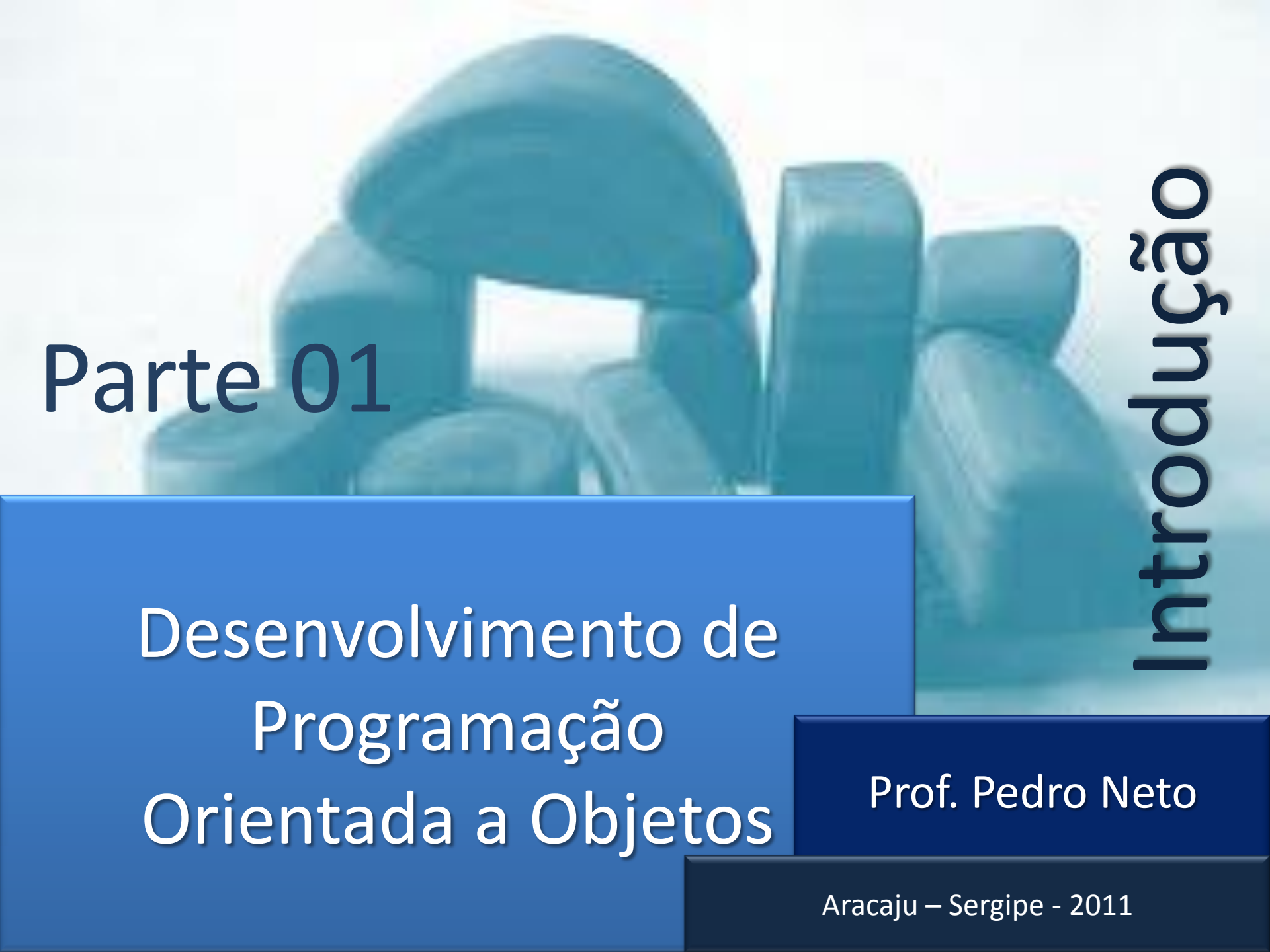


Introdução

Parte 01

# Desenvolvimento de Programação Orientada a Objetos

Prof. Pedro Neto

Aracaju – Sergipe - 2011

# Conteúdo

1. Introdução
    i. Paradigmas de Programação
    ii. Motivação da OO
    iii. Desafio das novas tecnologias
    iv. Ambientes de Desenvolvimento Modernos
    v. Programação OO

# Linguagens de Programação

*"O que caracteriza uma linguagem de programação?"*

- Sintaxe e semântica bem definidas;
- Implementável (executável) com eficiência aceitável.
- Universal: deve ser possível expressar todo problema computável.
- Natural para expressar soluções de problemas (em um certo domínio de aplicação)

# Paradigmas de Programação

## Linguagens de Programação


*"Por que existem tantas?"*

- Propósitos diferentes;
- Avanços Tecnológicos;
- Aderência a interesses de negócio;
- Cultura e background científico.

Paradigmas de Programação

# Definição de Paradigma

"Modelo, Norma ou Padrão de comportamento a serem seguidos"

"Paradigma é a representação do padrão de modelos a serem seguidos. É um pressuposto filosófico matriz, ou seja, uma teoria, um conhecimento ..."

"Um modelo ou exemplo; as inflexões de uma palavra; um modo de se pensar sobre, encarar algo ou agir de acordo com algum contexto; um sistema de regras e regulamentos (que podem ser escritos ou não) que faz duas coisas: 1) estabelece fronteiras ou limites; 2) determina como se comportar dentro..."

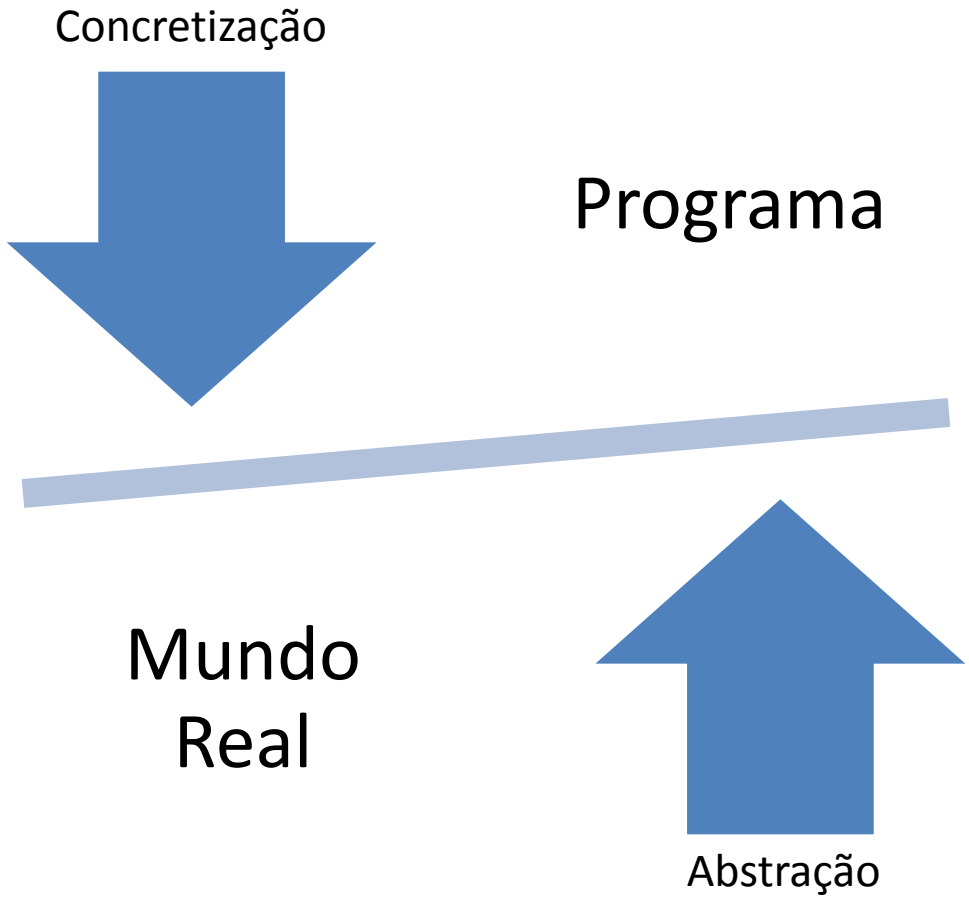
# Paradigmas de Programação

## Definição de Paradigma de Programação

*"Um **paradigma de programação** fornece e determina a visão que o programador possui sobre a estruturação e execução do programa." (Wikipedia)*

Determina a maneira como um programador abstrai um modelo do mundo real para uma linguagem de programação.

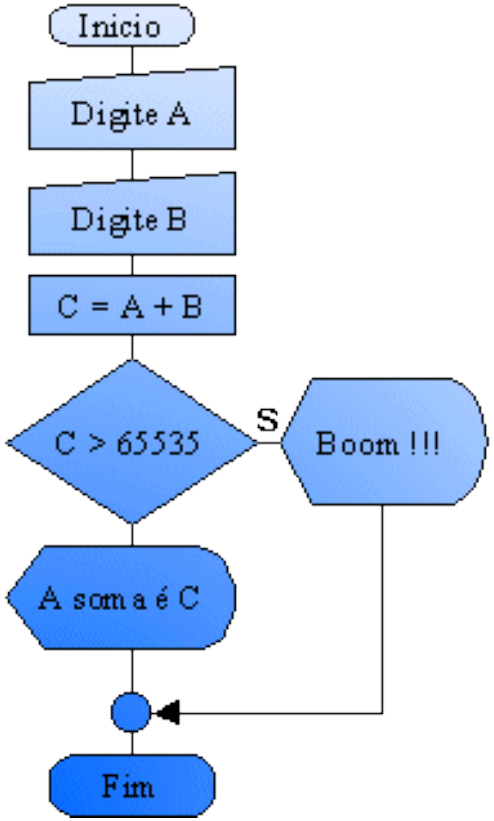
# Paradigmas de Programação



# Paradigmas de Programação

## Abstração, exemplo:

```
Write ('Entre A');
ReadLn (A);
Write ('Entre B');
ReadLn(B)
C := A + B;
If C > 65535 Then
   writeLn ('Boom !!!')
else
  writeLn ('A soma é C',C);
```




Um problema do mundo real é descrito em termos de variáveis (nome, tipo, endereço), ciclos, condicionais, atribuições, expressões (valor, tipo), entrada e saída e comandos

# Paradigmas de Programação

## Paradigma Imperativo/Procedural

- Programas centrados no conceito de um estado (modelado por variáveis) e ações (comandos) que manipulam o estado
- Inclui sub-rotinas ou procedimentos como mecanismo de estruturação
- Primeiro paradigma a surgir e ainda é o dominante

# Paradigmas de Programação

## Paradigma Imperativo/Procedural

# Paradigmas de Programação

## Paradigma Imperativo/Procedural

**Vantagens**

- Eficiência (embute modelo de Von Neumann)
- Modelagem “natural” de aplicações do mundo real
- Paradigma bem estabelecido

**Problemas**

- Relacionamento indireto entre E/S resulta em:
- Difícil legibilidade
- Erros introduzidos durante manutenção
- Descrições demasiadamente operacionais focalizam o “como” e não o “quê”.

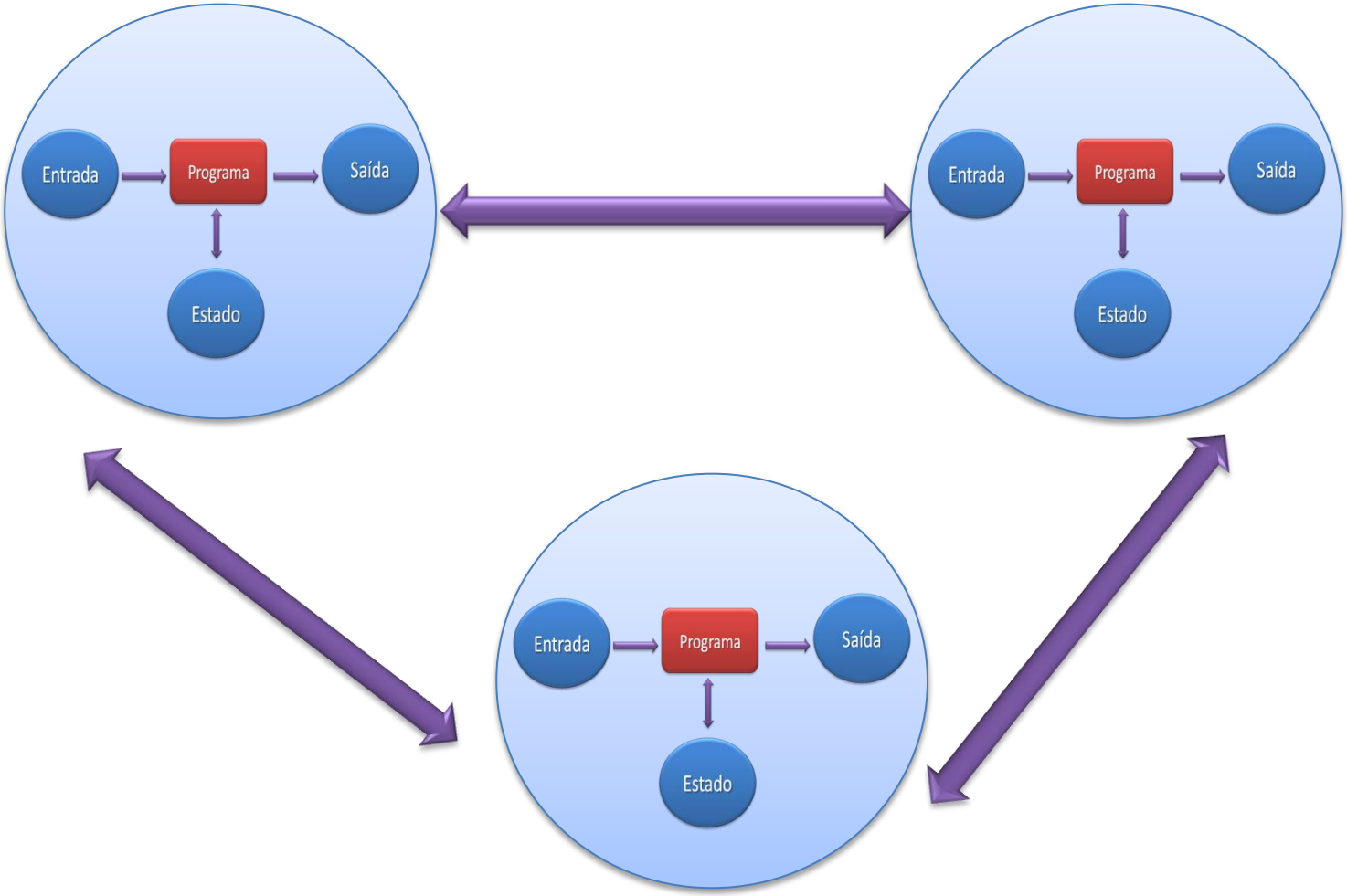
# Paradigmas de Programação

## Paradigma Orientado a Objetos

- Sub-classificacão do Paradigma Imperativo, onde a diferença é mais de metodologia quanto à concepção e modelagem do sistema.
- Em outras palavras, uma aplicação é estruturada em módulos (classes) que agrupam um estado e operações (métodos) sobre este.
- Classes podem ser estendidas e/ou usadas como tipos (cujos elementos são objetos)

# Paradigmas de Programação

## Paradigma Orientado a Objetos



# Paradigma Orientado a Objetos

**Vantagens**

- Todas as do estilo imperativo
- Classes estimulam projeto centrado em dados.
- Modularidade, reusabilidade e extensibilidade
- Aceitação comercial crescente

**Problemas**

- Semelhantes às do paradigma imperativo, mas amenizados pelas facilidades de estruturação

# Outros Paradigmas

**Funcional**

- Programas são funções que descrevem uma relação explícita e precisa entre E/S
- Estilo declarativo: não há o conceito de estado nem comandos como atribuição

**Lógico**

- Programas são relações entre E/S
- Estilo declarativo, como no paradigma funcional
- Na prática, inclui características imperativas, por questão de eficiência

# Paradigmas de Programação

## Outros Paradigmas

### Funcional


### Lógico

# Outros Paradigmas

**Funcional e Lógico**

**Vantagens**

- Manipulação de programas mais simples:
  - Prova de propriedades
  - Transformação (exemplo: otimização)
- Alto nível de abstração (E/S)

**Problemas**

- O Mundo não é funcional
- Dificuldade de abstrair problemas do mundo real

# Outras Classificações

## Quanto à Geração:

- **Primeira Geração**, as linguagens de baixo nível (Assembly)
- **Segunda Geração**, as primeiras linguagens (Fortran, ALGOL, ...)
- **Terceira Geração**, as procedurais e estruturadas (Pascal, C).
- **Quarta Geração**, linguagens que geram programas em outras linguagens (Java, C++), linguagens de consulta (SQL).
- **Quinta Geração**, linguagens lógicas (Prolog).

# Outras Classificações

## Quanto ao grau de abstração

- **Linguagem de programação de baixo nível**, cujos símbolos são uma representação direta do código de máquina que será gerado, onde cada comando da linguagem equivale a um "opcode" do processador, como Assembly
- **Linguagem de programação de médio nível**, que possui símbolos que podem ser convertidos diretamente para código de máquina (goto, expressões matemáticas, atribuição de variáveis), mas também símbolos complexos que são convertidos por um compilador. Exemplo: C, C++
- **Linguagem de programação de alto nível**, composta de símbolos mais complexos, inteligível pelo ser humano e não-executável diretamente pela máquina, no nível da especificação de algoritmos, como Pascal, Fortran, ALGOL e SQL

# Motivação da Orientação a Objetos

*"No início da era dos computadores, o hardware era tão caro que o custo do software era, muitas das vezes, desprezado nas organizações. As aplicações eram também, se comparadas com a atualidade, representativamente mais simples. Com o passar do tempo observou-se que os custos associados ao hardware diminuíram gradativamente, enquanto que os custos do software aumentaram assustadoramente. Hoje, em uma instalação, os custos associados com o software superam em várias vezes os custos de hardware, e a tendência é que isto aumente cada vez mais." (Kamienski)*

# Porque utilizar a Orientação a Objetos

## 1º - ATENDER A NATUREZA DAS NOVAS APLICAÇÕES

- **Tamanho e Complexidade** – Pacotes, Integrações, Camadas, Paralelismo, Transações, Processamento Gráfico
- **Alta Confiabilidade –** Proporcional ao aumento da complexidade
- **Alto Desempenho –** Devido ao uso de novas tecnologias, o desempenho é fator crítico

Desafio das Novas Tecnologias

# Porque utilizar a Orientação a Objetos

**2º - ATENDER AO DESAFIO DAS NOVAS TECNOLOGIAS DE DESENVOLVIMENTO**

- **Construir Rápido**
- **Construir Barato**
- **Construir Flexível**

# Ambientes de Desenvolvimento Modernos

## Características

- Programação Visual
- Programação dirigida por Eventos
- **Programação Orientada a Objetos**
- RAD – Rapid Application Development
- Prototipagem
- Aplicações Distribuídas
- Utilizações de Padrões Abertos
- Integração com Processos e Ferramentas de Desenvolvimento (Gerência de Projetos, Gerência de Configuração, Testes)
- MDA – Modelo Driven Architecture
- POA – Programação Orientada a Aspectos

# Programação Orientada a Objetos

*"Programação orientada a objetos (POO) é uma metodologia de programação adequada ao desenvolvimento de sistemas de grande porte, provendo modularidade e reusabilidade. A POO introduz uma abordagem na qual o programador visualiza seu programa em execução como uma coleção de objetos cooperantes que se comunicam através de mensagens." (Kamienski)*

# Programação Orientada a Objetos

*"A Orientação a Objetos surgiu com a necessidade de se criar um paradigma de programação simples, baseado na percepção humana dos objetos ao seu redor. Este novo paradigma não é apenas um modo de programar, mas uma maneira de pensar e conceber idéias. O software é composto por uma coleção de objetos que interagem entre si através de mensagens, simulando ações que ocorrem no mundo real." (Oliveira et al.)*

# Características

- Usa Objetos, e não Funções ou Procedimentos como seu bloco lógico fundamental de construção de programas
- Objetos comunicam-se através de mensagens
- Cada objeto é instância de uma Classe
- Classes podem se relacionar com outras (Herança, Composição, Agregação)

# Histórico

- **1961**
  - Nygaard e Dahl
  - Simula0, Simula1, Simula67
- **1970**
  - Research Center Xerox, Palo Alto
  - Smalltalk
- **1985**
  - Bertrand Meyer
  - Eiffel

# Histórico

- **1986**
  - Bjarne Stroustrup
  - C++
- **1996**
  - Gosling
  - Java
- **Atualmente**
  - .NET, PHP, Object Pascal, Objective C.

# Procedural X Orientado a Objetos

*"A programação orientada a objetos dá ênfase à estrutura de dados, adicionando funcionalidade ou capacidade de processamento a estas estruturas. Em linguagens tradicionais, a importância maior é atribuída a processos, e sua implementação em subprogramas. Em uma abordagem procedural, procedimentos ativos agem sobre dados passivos que foram passados a eles. Em linguagens orientadas a objetos, ao invés de passar dados a procedimentos, requisita-se que objetos realizem operações neles próprios." (Kamienski)*

# Vantagens do uso da Orientação a Objetos

- Prover mecanismos para visualizar a complexidade do desenvolvimento de software da mesma forma que visualizamos a complexidade do mundo real
- Acelerar o desenvolvimento de softwares com base na modularidade e acoplamento
- Melhorar a qualidade do software desenvolvido

# Dados de Contato

79 9949 4098

http://pedroneto.yolasite.com/

pedro@pyxistec.com.br

psneto@emsergipe.com

pedro.pyxistec@gmail.com

http://www.facebook.com/pedro.neto.se

pedropyxis

http://lattes.cnpq.br/4891420246888248