**Introdução ao paradigma de programação**

**Paradigma**

Uma ideia que se enquadra numa, tem métodos que devem ter formatos e respeitados e devem ser padronizados. Regem num modelo que vai focar o contexto e obedecer o padrão.

Num paradigma deve haver:

* Reutilidade: re-utilizacao do codigo
* Integridade
* Confidencialidade
* Confiabilidade: confiar o remetente a informacao que esta a ser entregue

1. O que e paradigma?

Um paradigma e o que determina o ponto de vista da realidade e como se actua sobre ela, os quais, são classificados, quanto ao seu conceito de base, podendo ser: imperativo, funcional, logico, orientado a objectos e estruturada. Cada qual determina uma forma particular de abordar os problemas e de formular respectivas soluções. Alem disso, uma linguagem de programação pode combinar dois ou mais paradigmas, para potencializar as analises e soluções.

Um paradigma pode ser concebido como um modelo, um padrao, contendo ideias claras, métodos eficientes de formas exactas**; E um modelo interactivo de uma realidade, que permite organizar as ideias com vista ao atendimento desta realidade e a determinação de qual e a melhor forma de actuar sobre essa realidade.**

Um paradigma pode ser concebido como um modelo, um padrao ou estilo de programação suportado por linguagens que agupam certas características comuns.

**Uma linguagem de programação e um método padronizado para expressar instruções para um computador.** E um conjunto de regras sintácticas e semânticas usadas para definir um programa de computador.

Regras sintácticas exprimem problemas para encontrar uma ideia clara e exacta.

Regra semântica: ortografia, sintaxe regar para poder declarar um atributo, método, etc.

N.B.: Deste modo cabe ao programador, escolher o paradigma mais adequado para analisar e resolver cada problema.

**Cateréticas de paradigmas de programação**

* Gramatica e significado bem definidos (sintaxe: gramatica (forma), semântica: significado);
* Implementável (executável), com eficiência **‘aceitável’**;
* Universal, deve ser possível expressar todo problema computável.

**Historial de paradigmas de programação**

A primeira linguagem de programação para computadores foi provavelmente Plankalkul criada por Konrad Zuse na Alemanha nazista, mas que teve pouco ou nenhum impacto no futuro das linguagens de programacao.

A primeira linguagem de programacao de alto nível altamente usada, foi o fortran, 1954.

1. Porque estudar linguagens de programação?

Programacao e central para computação;

* Linguagens de consulta a base de dados têm muitos dos conceitos de linguagens de programação;
* Linguagens de comandos de sistemas operacionais tem muitos dos conceitos de linguagens de programação;
* Linguagens de descrição de hardware tem muitos dos conceitos de linguagens de programacao;
* Processamento de linguagem natural e relacionado a processamento de linguagem de programacao;

1. Para que servem as linguagens de programacao?

Elas permitem que programadores tenham uma maior produtividade, ajudando os a expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de maquina).

* Tornam os programadores menos dependentes de computadores ou ambientes computacionais específicos.

**Propriedades desejáveis de uma linguagem de programação**

1. Legibilidade
2. Regibilidade vem de registo
3. Confiabilidade
4. Eficiência
5. Facilidade de aprendizado
6. Ortogonalidade, tem haver com a calegrafia
7. Re-usabilidade
8. Modificabilidade
9. Portabilidade
10. Integridade
11. Porque tantas linguagens de programação

Propósitos diferentes, avanços tecnológicos, interesses comerciais, cultura e background cientifico.

1. **Paradigma imperativo-conceitos**

O paradima imperativo e baseado na arquitectura de Von Neumann (foi o autor da arquitectura de computador) e o primeiro paradigma a existir e ate hoje e o dominante. Este paradima segue o conceito de um estado e de acções que manipulam esse estado, nele encontramos procedimentos que seguem de mecanismos de estruturação.

Ou seja organiza instruções de forma estruturada

Arquitetura de Neumann, caracteriza se por uma maquina digital armazenar seus programas no mesmo espaço de memoria que os dados. Um computador usa o CPU e a memoria para processar e armazenar instruções e dados.

Podemos denomina lo de procedual por incluir sub rotinas ou procedimentos de estruturação.

E o paradigma **mais utilizado**. Porque e com base nele que a logica de programação tem o se significado, coerência, modelo de organizar as instruções.

Exemplo de LP (linguagem de programação que baseiam no paradigma imperativo:

ALDA, ALGOL, ACEMBLER, BASIC, C, COBOL, FORTRAN, PASCAL, PHYTON, LUA.

Vantagens do paradigma imperativo

* Eficiencia (embute o modelo de Neumann);
* Modelagem “natural” de aplicações do mundo real, paradigma dominante e e tambem muito flexível

Desvantagens

* Dificil legibilidade;
* Erros introduzidos durante a manuntencao
* Descricoes demasiadamente operacionais focalizam o **como** e não **o que**

1. **Paradigma estruturado**

Este paradigma preconiza que todos os programas possíveis podem ser reduzidos a apenas 3 estruturas: sequencia (logica, ordem, as sequencias estão orientadas a uma solucao), decisão (condições existentes) e interacção (ciclo de repetição, de todas condições existentes trazer apenas aquela que agregam valor, ou que eu precise).

Tendo, na pratica sido transformada na programação modular, a programacao estruturada orienta os programadores para a criação de estruturas simples em seus programas usando as sub-rotinas e as funções. Foi a forma dominante na criação de software entre a programação linear e a programação orientada por objectos.

Apesar de ter sido pela POO, pode se dizer que a programacao estruturada ainda e marcantemente influente, uma vez que grande parte das pessoas ainda aprendem programacao através dela.

Este paradigma e mais usual para o aprendizado.

C, BASIC, PASCAL e COBAL.

Vantagens do paradigma estruturado

* Os problemas podem ser quebrados em vários sub problemas;
* A boa legibilidade e a boa compreensão da estrutura deste paradigma motivam os programadores a iniciarem a programacao pelo modelo estruturado.

Desvantagens

* Os dados são separados das funcoes;
* Mudancas na estrutura dos dados acarreta alterações em todas as funcoes relacionadas;
* Gera sistemas difíceis de serem mantidos;

1. **Paradigma orientado a objectos**

O paradigma orientado a objecto e baseado na composição e iteração de diversas unidades de softwares denominados objectos. O funcionamento de um software orientado a objectos se da através do relacionamento e troca de mensagens entre estes objectos. Estes objectos são classes e nessas classes os comportamentos são chamados de **métodos** e os estados possíveis da classe são chamados de **atributos.** Nos métodos e nos atributos também são definidas as formas de relacionamento com outros objectos.

* Não e um paradigma no sentido restrito, e uma sub classificação do imperativo;
* A diferenca e mais de metodologia quanto a concepção e modelagem do sistema;
* A grosso modo, uma aplicação estruturada em módulos (classes que agrupam um estado (atributos) e operações (métodos) sobre este);
* Classes podem ser estendidas e/ou usadas como tipos (cujos elementos são objectos).

O modelo computacional do paradigma orientado a objectos obedece uma sequencia a partir dos dados de entrada, um processamento do programa através do seu estado apresentando a posterior a saída.

Vantagens do paradigma orientado a objectos

* Eficiencia (embute o modelo de Neumann);
* Modelagem “natural” de aplicações do mundo real, paradigma dominante e e tambem muito flexível
* Classes estimulam projectos centrados em dados: modularidade, re-usabilidade e extensibilidade;
* Aceitacao comercial crescente.

Problemas encontrados no paradigma orientado a objectos

São semelhantes a o do paradigma imperativo mas são amenizados pelas facilidades de estruturação.

Exemplo de linguagens: ADA, ALGOL, BASIC, C, COBAL E FORTRAN.

Desvantagens

* Por exigir formas de pensar relativamente complexas a POO no seu paradigma e ate hoje ainda não bem compreendida ou usada pela maioria.

1. **Paradigma Paralelo e distribuído**

**Panorama actual**

1. O poder de processamento das máquinas vem crescendo rapidamente
2. Grande parte das máquinas são interligadas por redes de computadores.

Dai que a programação paralela e distribuída a nível comportamental, os seus sistemas e aplicações estão cada vez mais complexos: funcionalidade, interfaciamento gráfico, comunicação; maior carga, maior número de usuários; melhor tempo de resposta, maior confiabilidade.

**Paradigma Paralelo**

**Conceito**: consiste em executar simultaneamente várias partes de uma mesma aplicação;

* Tornou se possível a partir do desenvolvimento de sistemas operacionais multitarefas, multitaread e paralelos.

**Aplicações são executadas paralelamente devido a:**

* Um mesmo processador;
* Em uma máquina multiprocessada;
* Em um grupo de máquinas interligadas que se comportam como uma só máquina.

**Paradigma Distribuído**

Conceito: consiste em executar aplicações operantes em máquinas diferentes. Tornou se possível a partir da popularização das redes de computador.

**Aplicações são executadas em maquinas diferentes interligadas por uma rede**, **que consistem em:**

* Intranet, internet e outras redes públicas ou privadas.

**Diferença entre paradigma paralelo e paradigma distribuído**

1. **Quanto ao acoplamento:**

* Os sistemas paralelos são fortemente acoplados: compartilham hardware ou se comunicam através de um barramento de alta velocidade. Os sistemas distribuídos são fracamente acoplados.

1. **Quanto a previsibilidade:**

* Os comportamentos dos sistemas paralelos é mais previsíveis enquanto que os sistemas distribuídos são mais imprevisíveis devido ao uso da rede e a falhas

1. **Quanto a influência do tempo:**

* Os sistemas distribuídos são bastantes influenciados pelo tempo de comunicação pela rede, que no geral não há uma referência do tempo global;
* Em sistemas paralelo o sistema de troca de mensagem pode ser desconsiderado.

1. **Quanto ao controle:**

* No geral em sistemas paralelos se tem o controlo de todos os recursos computacionais enquanto que nos sistemas distribuídos tendem a empregar recursos de terceiros.

**Vantagens**

* Usam melhor o poder de processamento;
* Apresentam um melhor desempenho;
* Permitem compartilhar dados e recursos;
* Podem apresentar maior confiabilidade;
* Permitem reutilizar serviços já disponíveis;
* Atendem um maior número de usuários.

**Dificuldades**

* Desenvolver, gerenciar e manter o sistema;
* Controlar o acesso concorrente a dados e recursos compartilhados;
* Evitar que falhas de máquinas ou da rede comprometam o funcionamento do sistema;
* Garantir a segurança do sistema e o sigilo de dados trocados entre máquinas;
* Lidar com a heterogeneidade do ambiente.

**Plataformas de Execução**

Um sistema operativo multitarefa permite simular o paralelismo em um único computador, alterado a execução de processos.

Um processador com núcleo único permite paralelismo real entre processos, executando múltiplas instruções por círculo.

Uma placa mãe multiprocessador permite cada processador execute um processo.Um clistor e uma solução de baixo custo pra processamento de auto desempenho.

1. **Paradigma Funcional**

Tem origem a linguagem funcional que é um paradigma de programação baseada em funções matemáticas ou Isto é, **este paradigma trata a computação como uma avaliação de funções matemáticas.**

Este método enfatiza aplicação de funções, as quais são tratadas como valores de primeira importância, ou seja, funções podem ser parâmetros ou valores de entrada para outras funções e podem os valores de retorno ou saída de uma função.

O paradigma funcional tem sido mais usado academicamente do que no desenvolvimento de software. O interpretador da linguagem funcional atua como uma calculadora: **lê, calcula e mostra o resultado**.

**Vantagens do paradigma funcional**

As linguagens funcionais fornecem um alto nível de abstração, o que faz com que os programas funcionais sejam mais pequenos, claros e rápidos.

Devido ao processo automático de alocação de memória então os efeitos colaterais no cálculo da função são eliminados. Sem esses efeitos, a linguagem assegura que o resultado da função será o mesmo para um dado conjunto de parâmetros não importando onde ou quando seja avaliada que é empregado em computações independentes para execução paralela. A recursividade em programação funcional pode assumir varias formas e no geral uma técnica mais poderosa que o uso de laços do paradigma imperativo.

**Desvantagens**

Na programação funcional parece faltarem diversas construções (embora corretamente) consideras essências em linguagens imperativas como C.

Por exemplo não alocação explícita de memória nem de variáveis.

Os programas funcionais podem ser menos eficientes.

**Exemplos do paradigma funcional**: LISP, ML, MIRANDA E HASKELL;

Lisp - foi a primeira linguagem de programação funcional, criada por Jonh Mascharthy no fim dos anos 50;

HASKEL – Surgiu no fim dos anos 80.

Usa o workflow-diagrama de processos, acomoda requisitos funcionais;

1. **Paradigma logico**

O que determina a escolha de um paradigma?

E o contexto da resolução, o problema identificado, ver num problema qual e o melhor paradigma para sua resolução, mesmo escolhermos um podemos escolher outro de forma interna.

Paradigma concorrente

A programação concorrente e um paradigma de programação usado na construção de programas que fazem o uso da execução simultânea de diversas tarefas que podem ser implementadas como programas separados ou como um único programa que dispara varias linhas de execução em paralelo. As linhas de execução tambem são conhecidas como **threads.**

A programação concorrente permite a execução em paralelo de vários programas no computador e esta relacionada com a programação paralela.

As tarefas em linha de execução podem ser concorrentemente em um único processador, vários processadores em um único equipamento ou vários processadores espalhados em uma rede de servidores.

Vantagens

A principal vantagem do uso da programação concorrente e o aumento do desempenho dos programas, pois e, possível aumentar a quantidade de tarefas executadas em um determinado período de tempo. Por ex.: em uma grande empresa, pode se calcular a facturação mensal da empresa executando o calculo da facturação de forma concorrente processando vários filiais simultaneamente.

Neste caso a programação concorrente permite ganho de tempo pois processamos todas as filiais ao mesmo tempo, ao invés de processar a facturação de uma filial de cada vez. Esta abordagem e muito usada profissionalmente.

Outra vantagem e que o projecto dos programas se torna mais simples com a relação a concorrência entre os programas pelos recursos compartilhados.

Desafios

O grande desafio dessa programação, e a partilha de recursos que são executados concorrentemente. O objectivo deve ser o de administrar o acesso concorrente aos recursos computacionais. Tais como por ex.: os acesso aos discos ou a outros recursos compartilhados com a impressora.

Comunicacao e interacao entre programas concorrentes

Em um sistema operativo concorrente (computacional), os programas executados de forma concorrente podem ter a necessidade de se comunicarem para a troca de informações. Neste contexto e importante que o programar tenha que ter alguma informação sobre como e feita a comunicação entre os programas executados concorrentemente. Esta informação sobre a comunicação entre a comunicação entre programas concorrentes e importante para que se evite que um programa interfira em outro programa.

Conflitos no compartilhamento de recursos

**Introdução**

A programação concorrente é um paradigma de programação usado na construção de programas que fazem uso da execução simultânea de diversas tarefas que podem ser implementadas como programas separados ou como um único programa que dispara várias linhas de execução em paralelo. As linhas de execução também são conhecidas como threads.

A programação concorrente permite a execução em paralelo de vários programas no computador e está relacionada com a programação paralela.

As tarefas sendo podem ser executadas concorrentemente em um único processador, vários processadores em um equipamento ou vários processadores espalhados em uma rede de servidores.

**Vantagens da Programação Concorrente**

A principal vantagem do uso da programação concorrente é o aumento do desempenho dos programas, pois é possível aumentar a quantidade de tarefas executadas em um determinado período de tempo.

Por exemplo, em uma grande empresa, pode-se calcular o faturamento mensal da empresa executando o cálculo do faturamento de forma concorrente, processando várias filiais simultaneamente. Neste caso, a programação concorrente permite ganho de tempo pois processamos todas as filiais ao mesmo tempo, ao invés de processar o faturamento de uma filial de cada vez. Esta abordagem é muito utilizada profissionalmente.

Outra vantagem é que o projeto dos programas se torna mais simples com relação à concorrência entre os programas pelos recursos compartilhados.

Desafios da Programação Concorrente

O grande desafio da programação concorrente é o compartilhamento de recursos, a comunicação e a interação entre os programas que são executados concorrentemente. O objetivo deve ser o de administrar o acesso concorrente aos recursos computacionais, tais como por exemplo, o acesso aos discos ou a outros recursos compartilhados, como a impressora.

Comunicação e Interação entre programas concorrentes

Em um sistema computacional concorrente, os programas executados de forma concorrente podem ter a necessidade de se comunicarem para troca de informações. Neste contexto, é importante que o programador possa ter alguma informação sobre como é feita a comunicação entre os programas executados concorrentemente. Esta informação sobre a comunicação entre programas concorrentes é importante para que se evite que um programa interfira em outro programa.

Conflitos e Compartilhamento de Recursos

É preciso evitar conflitos entre os programas concorrentes. Os conflitos podem ocorrer porque os programas concorrentes fazem uso de recursos compartilhados e por isso a necessidade de se ter controle de concorrência.

Em alguns sistemas computacionais concorrentes a comunicação entre os processos concorrentes é 'escondida' do programador mas em outros sistemas a comunicação é explícita, podendo ser interpretada pelo programador. Existem dois tipos de comunicação explícita para a programação concorrente: a comunicação por memória compartilhada e a comunicação por troca de mensagens.

**Comunicação por troca de mensagens**

Este tipo de comunicação consiste na troca de mensagens entre processos sendo executados concorrentemente através da chamada de funções ou serviços. Caso os processos estejam sendo executados em máquinas diferentes é a unica forma de comunicação possível. Baseia-se na colaboração entre os processos, onde para uma mensagem enviada por um processo sempre deve haver uma resposta pelo outro processo.

**Comunicação por memória compartilhada**

Este tipo de comunicação os processos comunicam-se através do acesso à áreas de memória que são compartilhadas entre os processos concorrente. Neste tipo de comunicação deve ser feiro gerenciamento do uso da memória compartilhada a fim de que um processo não interfira na execução de outro processo sendo executado concorrentemente.

A comunicação por troca de mensagens é mais simples e mais eficiente do que a comunicação por memória compartilhada.

**Suporte à programação concorrente**

São poucas as linguagens com suporte nativo à programação concorrente. Actualmente apenas JAVA e C# possuem recursos que permitem a construção de programas concorrentes. Destas duas linguagens, JAVA se destaca por ter portabilidade a outros sistemas operacionais ao contrário de C# que é restrita aos sistemas Windows.

As linguagens C e C++ oferecem suporte à programação concorrente através de bibliotecas de funções para programação concorrente.

**Linguagens específicas para Programação Concorrente**

E existem ainda as linguagens que foram projetadas especificamente para programação concorrente. A linguagem Erlang é muito utilizada atualmente. A principal vantagem de Erlang é suporte a concorrência. Tem um pequeno, mas poderoso, conjunto de funções primitivas para criar processos e fazer com que eles se comuniquem. Processos são o principal meio de criar uma aplicação em Erlang.

Introdução

• Evolução das LPs

– Primeira Geração: Linguagem de máquina

• Código de Máquina (0s e 1s).

– Segunda Geração:

• Linguagem de Montagem - Assembler

– Terceira Geração

• Imperativas: FORTRAN, Cobol, Basic, Algol, ADA, Pascal, C

• Lógicas e Funcionais: LISP, ML, Prolog

– Quarta Geração

• Geradores de Relatórios, Linguagens de Consultas: SQL, CSP

– Quinta Geração

• LOO : Smalltalk, Java, Eiffel, Simula 67

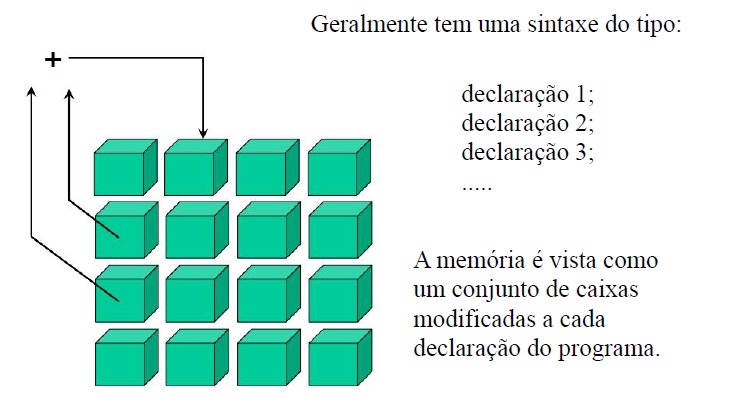
– Sexta Geração ?

• Web e Linguagens Dinâmicas : Python, JavaScript, Ruby

1. Linguagens Imperativas

Linguagens imperativas ou procedurais são linguagens orientadas à comandos ou declarações. Elas baseiam-se na idéia de que a memória mantém uma máquina de estados que pode ser modificada a cada execução das declarações da linguagem.

A Figura a seguir ilustra este processo.



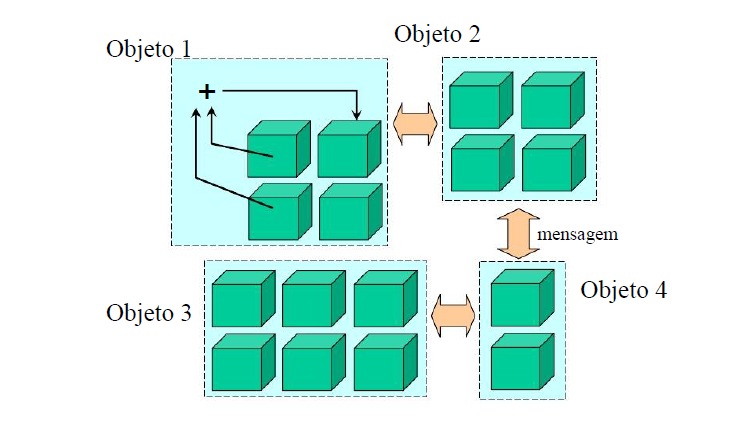
Desenvolvimento de programas consiste na modificação consecutiva dos estado da memória até se chegue a uma solução desejada.

Este paradigma representa geralmente a primeira abordagem que alguém aprende em programação (no MIT usava-se programação funcional como primeiro curso). Algumas das mais usadas linguagens de programação usam este modelo de programação:

C, FORTRAN, Algol, Pascal, COBOL, “C++”, “Java”, etc.

1. Linguagens orientadas a objecto

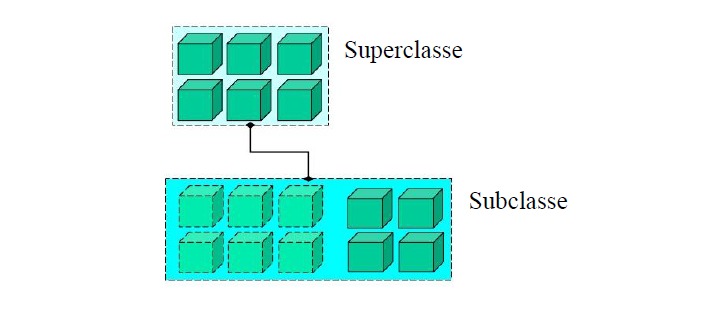
Linguagens orientadas à objecto são linguagens que estendem a linguagens imperativas com a introdução do conceitos de objetos. Objetos são conjuntos de dados geralmente complexos com funções bem definidas que podem ser aplicadas eles. Funções e dados são encapsulados pelo objeto de forma a expor somente o que se quer a outros objetos. Objetos se comunicam por mensagens, que chamam as funções expostas. Para facilitar esta prática, funções similares podem ter o mesmo nome deste que tenham estruturas diversas (polimorfismo em tempo de compilação), ou a associação entre uma mensagem e um objeto (e a função chamada) pode ser feita dinamicamente em tempo de execução (polimorfismo real).



Objetos são definidos a partir de outros objetos mais genéricos através de mecanismos de herança.

Subclasse

VIDE A FIGURA ABAIXO



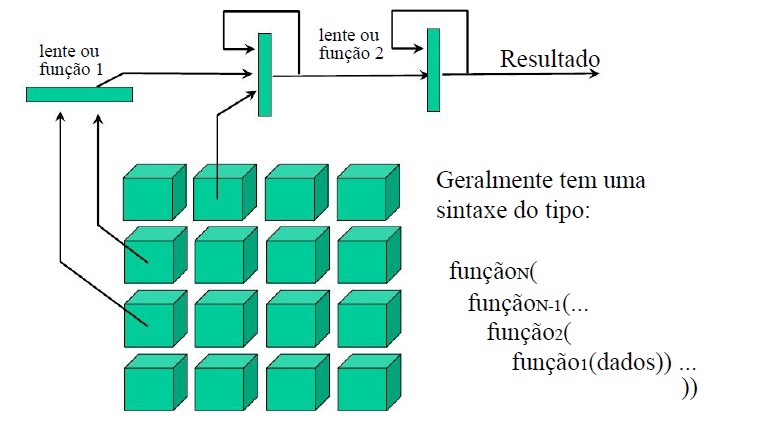
Ao construir objetos concretos na memória, OO usa a eficiência da programação imperativa. Ao construir classes de funções que aplicam-se a um conjunto restrito de objetos, OO ganha através de encapsulamento e polimorfismo a flexibilidade e confiabilidade da programação funcional (nossa próxima transparência).

São exemplos de linguagens OO as linguagens Smalltalk, Ada (não tem polimorfismo real), C++, e Java.

1. Linguagens Funcionais

Linguagens funcionais ou aplicativas são linguagens orientadas à função que um programa representa. Isto é conseguido pensando- se na função que deve ser aplicada num estado de máquina inicial para transformá-lo em um estado de máquina final desejado como resposta.

Pode-se ver este processo como a construção de uma lente que pega o estado inicial da memória e o “transforma” em um estado final desejado. A Figura a seguir ilustra este processo.



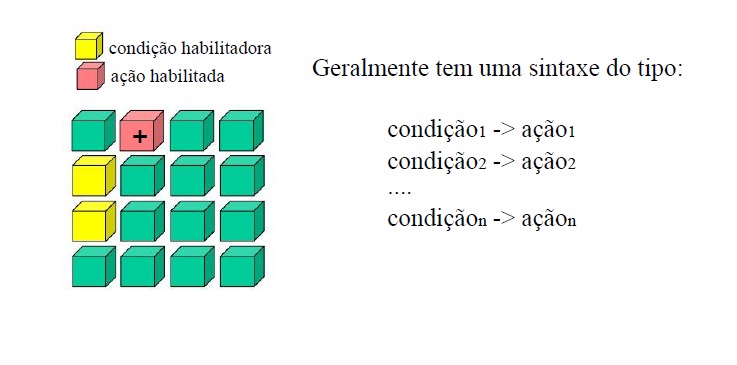
Desenvolvimento de programas consiste em sucessivamente desenvolver funções a partir de funções previamente existentes (bottom-up), até chegar uma função final (geralmente complexa) que consegue computar a resposta desejada a partir do conjunto inicial de dados.

Em vez de olhar para sucessivas máquinas de estados de uma computação, a programação aplicativa considera sucessivas transformações funcionais. LISP, ML e HASKELL são três da mais conhecidas linguagens funcionais.

1. Linguagens logicas

Linguagens lógicas ou baseadas em regras são linguagens baseada na lógica de predicados, onde uma série de regras são definidas para que o programa tome ações apropriadas para cada estado habilitador na memória do computador.

Pode-se ver este processo como a construção de uma série de filtros (as regras) que sucessivamente habilitam mudança de estados. A figura a seguir ilustra este processo.



Desenvolvimento de programas consiste em construir-se um conjunto de regras que trate (tenha condições habilitadoras e tome ações adequadas) todos os possíveis estados de iniciais do programa.

Prolog (linguagem de programação lógica) é o exemplo mais conhecido desta família de linguagens, mas qualquer qualquer linguagem ou ferramenta baseada em tabelas de decisões podem ser vistas como linguagens baseadas em regras.

Exemplos incluem ferramentas de parsing tais como YACC (Yet Another Compiler Compiler) ou JavaCC.

1. Introducao a POO

Programação imperativa separa (ou tende a separar) dados dos procedimentos usados para manipular estes dados.

Programação OO concentra-se definir os objectos dentro de um certo domínio com um conjunto “estado” e “comportamento”.

Conceitos Básicos em OO

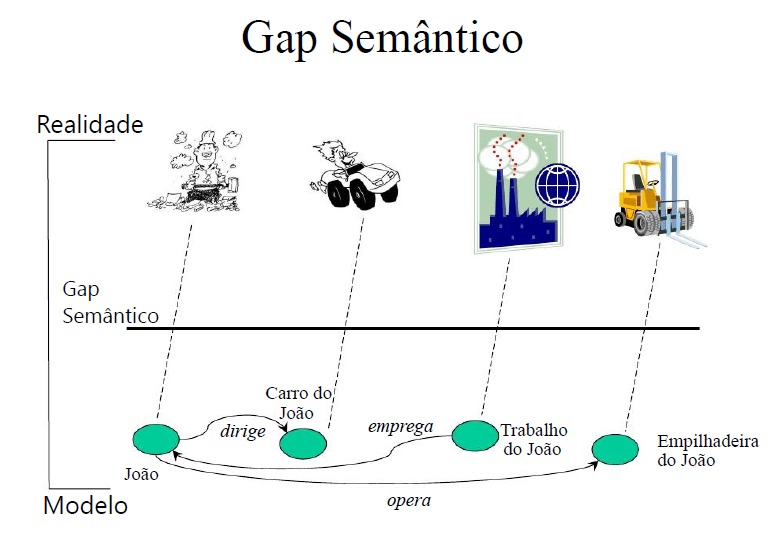
• Encapsulamento: os detalhes de implementação dos objetos são “escondidos” dos usuários deste objeto (só a sua interface e comportamento é de interesse real).

• Herança: novas classes de objetos podem ser criadas a partir de outras classes de objetos mais abstratos. Estes novos

objetos herdam e estendem as propriedades dos objetos mais abstratos.

• Polimorfismo (um nome muita formas): o remetente da mensagem não precisa se preocupar em saber os detalhes dos objetos destinatários desta mensagem. A mesma mensagem pode ser dinamicamente enviada para muitos tipos deobjetos.

VIDE O Gap Semântico



POO e o Gap Semântico

• O Gap Semântico é a diferença entre a forma como o modelo representa a realidade, e a realidade propriamente dita.

• Quanto menor o intervalo, mais fácil será a compreensão do sistema e a forma de alterá-lo. As alterações serão na maioria das vezes locais, afetando um ou poucos “indivíduos” que são representados por códigos contidos em objetos.

POO tenta reduzir o GAP Semântico !

Objetos

• Objeto: é uma entidade habilitada a ter um estado (informação) e oferece um determinado número de operações (comportamentos) que podem examinar ou afetar o estado do objeto.

• Modelo Orientado a Objeto: refere-se a modelos cujos componentes são representados por objetos.

Características da POO (1)

• Acesso a Informações: a única forma de acesso externo ao objeto será através dos métodos, suas particularidades internas devem ser protegidas do mundo externo.

• Encapsulamento: variáveis e métodos associados a um objecto são encapsulados. Esta é uma idéia simples e poderosa que proporciona duas vantagens:

Proteção dos Dados e Modularidade.

• Comunicação: a comunicação entre os objetos ocorre por Passagem de mensagens (chamada a um Método)

Tipo Abstrato de Dados encapsulam implementação e interface em uma classe de objetos. Uma ou mais instâncias de uma classe podem então ser instanciadas. Uma instância de uma classe é conhecida como um objeto. Todo objeto tem um estado e comportamento. Aonde o estado é determinado pelos valores atuais armazenados nas variáveis de instância e o comportamento é determinado pelos métodos de instância da classe da qual o objeto foi instanciado.

Classes e Instâncias

• Classe: é uma definição, um modelo existente para a criação de novos objetos. Pode ser considerada como uma abstração que descreve todas as características comuns dos objetos criados a partir dela.

• Usando o conceito de classe, características podem ser associadas a um grupo inteiro de objetos.

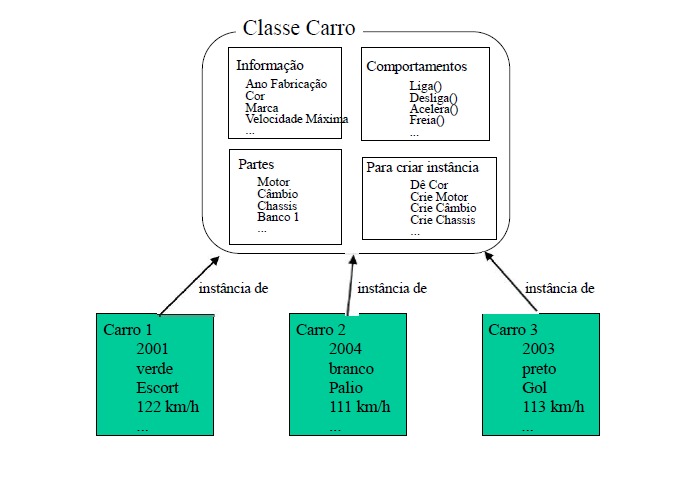
• Instância: um objeto que pertença a uma classe é chamado de instância desta classe.

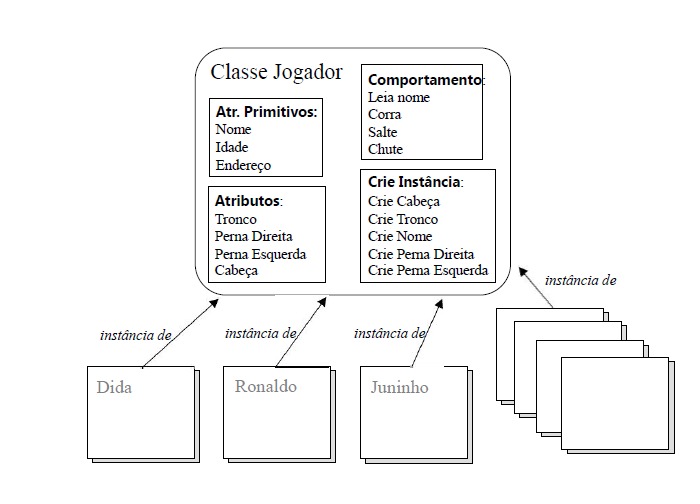
Mais Classes e Instâncias

• Uma classe é uma fôrma que descreve de forma genérica grupos de objetos com características similares.

• Uma instância de uma classe é um objeto real.

A classe representa a descrição do que seria um objeto (ex., um Carro que deve ter modelo, cor, ano, e dono) enquanto uma instância é uma representação concreta de um destes objetos na memória (ex., o Escort verde, ano 2001, do Prof. Manoel).





Atributos e Métodos

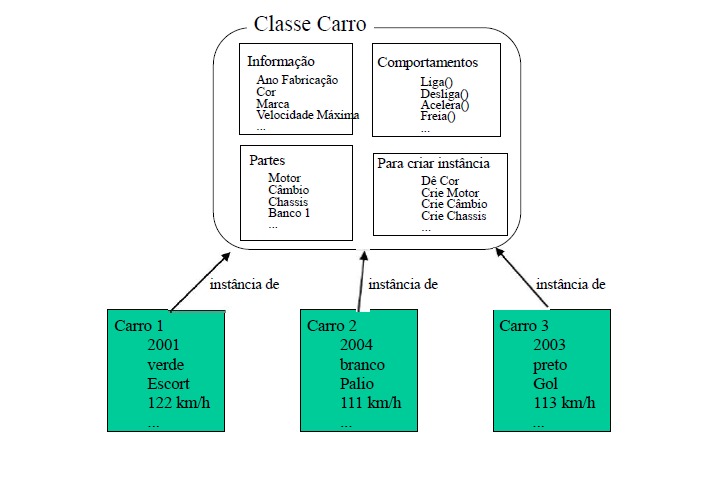
• Atributos

– As informações e partes que compõem o objeto

– Os valores dos atributos definem o estado do objeto

• Métodos

– Funções e Procedimentos que alteram o estado do objeto e modela o seu comportamento



Herança (1)

• Herança:

– Quando várias classes têm informações e comportamentos em comum, estes podem ser abstraídos para evitar redundância de implementação

– As informações e comportamentos comuns podem ser colocados em uma classe mais genérica, chamada superclasse

– As classes mais específicas podem então herdar estas informações e procedimentos da superclasse

Considere como exemplo a necessidade de se construir

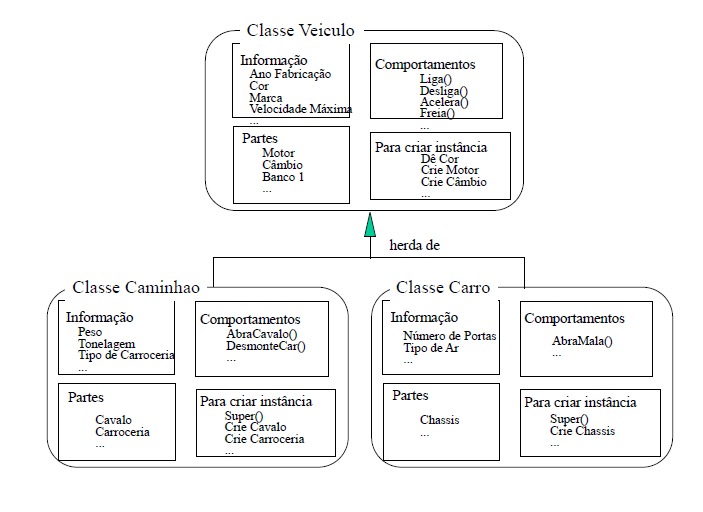
Caminhões e Ônibus, além de Carros. Os pontos em comum destas classes poderiam ser abstraídos para uma superclasse “Veículo”.

Herança (2)

• Através da herança, descrições comuns podem ser re-utilizadas, promovendo o conceito de código re-utilizável.

• Herança elimina a redundância pelo fato de classes descendentes apenas implementarem informações e comportamentos adicionais, que as diferenciam das demais. Isto conduz a sistemas menores e mais fáceis de compreender.

• Quando modificações são implementadas nas partes comuns (superclasses), todas as classes descendentes automaticamente herdam estas modificações. Isto possibilita a criação de modelos mais fáceis de se modificar.

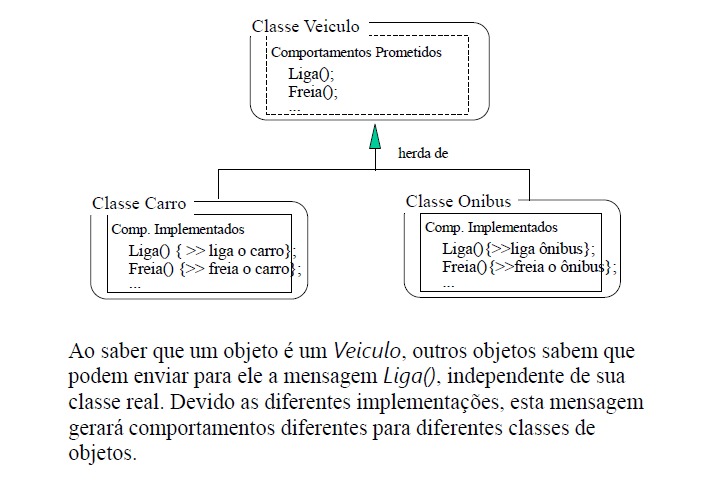


Polimorfismo

• superclasses também podem ser usadas para “prometer” um comportamento que as subclassesdevem implementar

• desta maneira o remetente de uma mensagem só precisa saber a um nível abstrato que o objeto tem aquele comportamento

• isto permite criar situações em que o comportamento real de um objeto é definido em tempo de execução



Exemplo de Polimorfismo

Considere que a variável “v” é definida com um veículo.

função ligueVeiculo( Veiculo v) {

v.liga();

}

Qualquer carro ou ônibus pode então ser associada a “v”:

Carro c1 = CrieCarro();

Onibus o3 = CrieOnibus();

ligueVeiculo(c1);

ligueVeiculo(o3);

Ambos veículos são ligados através da mensagem v.liga(), mas se as classes carro e ônibus têm implementações próprias para a rotina liga(). A mensagem “liga()” chamará rotinas diferentes quando “v” for um carro ou um caminhão.