Programador/a de informática

PSD

Princípios metodológicos de programação – 0806

Conteúdos

- Metodologias em programação local;
- Princípios do método;
- Estrutura de um programa;
- Corpo do programa;
- Desenho de formatos para entrada e saída dos dados em ecrã;
- Dicionário de dados.

Metodologias em programação local

- Evolução;
- Vantagens e desvantagens do seu uso;
- Tipos de metodologias;
- Desenho estruturado;
- Orientadas por objeto;
- Orientadas aos dados.

Evolução

- Em meados da primeira guerra mundial houve uma evolução significativa no segmento corporativo.
- Baseado nestas transformações houve a necessidade de se aplicar o conceito de dinamização de processos e daí surgiu à necessidade de se administrar grandes volumes de dados em organizações de todas as sectores.
- Com a criação dos computadores comerciais após a segunda guerra mundial houve um aumento significativo na dinamização da indústria de computadores e, consequentemente, o processo de construção de software, para que os mesmos automatizassem processos manuais e pudessem avaliar situações complexas que são parte integrante do quotidiano das organizações.

- A partir desse cenário, criaram-se modelos de desenvolvimento de software

Evolução

- O desenvolvimento de software deixou de ser sinónimo apenas de escrita de código, (sem metodologia);

- Hoje em dia é necessário utilizar uma metodologia de trabalho, isto é,

Um processo dinâmico e interativo para o desenvolvimento estruturado de um projeto, sistema ou sotfware, tendo em vista a qualidade e produtividade do mesmo.

É objetivo de uma metodologia definir de forma clara…

"quem" faz " o quê", "quando" , "como", e até mesmo "onde"

Para todos os que estejam diretamente envolvidos ou não com o desenvolvimento de software:

Técnicos, administradores, Analistas ou utilizadores

5

O PORQUÊ DA UTILIZAÇÃO DE MODELOS NA CONSTRUÇÃO DE SISTEMAS INFORMÁTICOS

- Existência de linhas de orientação que guiam o desenvolvimento do projeto:
  - Fornecem uma orientação normalizada;
  - Indicam quais as ferramentas certas de modo a que se possa executar com sucesso cada uma das tarefas.
- Promover a difusão da informação, relativas ao sistema, entre os indivíduos envolvidos na sua construção.
- Diminuição do risco de desperdício de recursos no desenvolvimento do sistema:
  - de erros é menos custosa quando detectada e realizada nos modelos do sistema (é mais fácil corrigir uma maqueta que deitar abaixo parte do edifício);
- Utilização de várias técnicas que se completam e validam entre si, e possibilitam uma visão mais ampla do sistema;
- Permitem predizer comportamentos futuros do sistema (os modelos servem como um "laboratório" onde diferentes soluções, para um problema relacionado com a construção do sistema, podem ser experimentadas)

6

Metodologias de desenvolvimento de software

Existem várias metodologias de desenvolvimento e cada uma tem as suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento.

Há no entanto atividades que, com uma ou outra modificação, são comuns à maioria das metodologias existentes:

Planeamento – determinar os custos , tempo e recursos necessários para a execução do projeto

Análise – investigação das necessidades, requisitos do sistema a implementar

Projeto – investigação proposta de solução com base nos dados adquiridos na análise

Codificação – Processo de criação do código dos diversos componentes do sistema

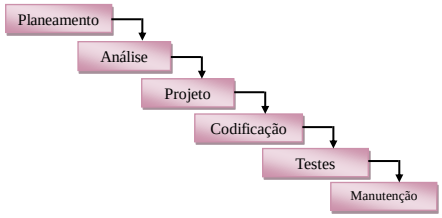
Testes – verificação se os componentes gerados atendem às especificações

Manutenção – Tempo de vida útil do sistema e durante o qual serão efetuadas todas as alterações após o início do funcionamento do sistema (correção de erros, introdução de melhorias/novas funcionalidades)

OUTRAS INTERVENÇÕES NO PROCESSO: estudo de viabilidade, controlo de qualidade, gestão do projeto e implantação.

Metodologia em cascata

- Modelo clássico em que as fases são executadas sequencialmente, de forma a que cada uma delas só tem início quando a anterior termina.
- No modelo em cascata passa-se para a próxima fase somente quando a fase anterior está completa e perfeita, isto é, só se avança para a tarefa seguinte quando o cliente valida e aceita os produtos finais (documentos, modelos, programas) da tarefa atual.
- Pressupõe que o cliente participa ativamente no projeto e que sabe muito bem o que quer.



Metodologia em cascata

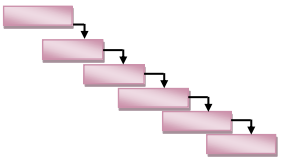
Vantagens

- método considerado simples, já que **impõe uma sequência restrita e bem dimensionada de tarefas.**

Desvantagens

- Se definido um requisito incorreto, a falha irá propagar-se a todas as outras fases subsequentes:
- Promove a divisão dos esforços ao longo das diferentes tarefas, logo desencoraja a comunicação e a partilha de visões entre todos os intervenientes do projeto;
- Como o processo não retrocede, é normal que as novas ideias não sejam reaproveitadas,
- Há uma grande extensão de período de tempo que decorre entre o início do projeto e a entrega do produto final.

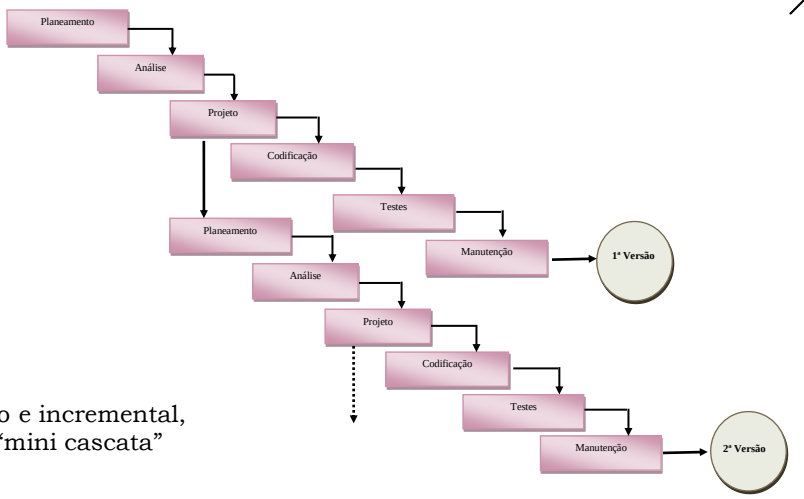
Por todas estas razões o modelo em cascata é visto como um exemplo de um método de risco e um convite a falhas



Metodologia Iterativa e incremental

- A noção de processo iterativo corresponde à ideia de "melhorar pouco a pouco" o sistema. O âmbito do sistema não é alterado, mas o seu detalhe vai aumentando em iterações sucessivas.
- Nele o sistema de software é desenvolvido em vários passos similares (*iterativo*). Em cada passo, o sistema é estendido com mais funcionalidades *(incremental*).
- O desenvolvimento de um sistema é dividido em ciclos; Em cada ciclo são abordadas todas as fases do processo de desenvolvimento;
- Cada iteração considera, em função do seu grau de importância, um subconjunto de requisitos a modelar;
- A cada nova iteração novos requisitos poderão ser considerados e os anteriores refinados, completando ao longo do tempo os produtos finais de todo o processo, mas em que cada iteração, por si, produz sempre um conjunto de produtos finais.

Metodologia Iterativa e incremental



No processo iterativo e incremental, cada ciclo é uma "mini cascata"

Metodologia Iterativa e incremental

Vantagens

- Os requisitos mais importantes e/ou de maior risco são considerados no início, nas primeiras iterações, sendo identificados os erros de maior impacto quando é possível e mais fácil tomar medidas para os corrigir;
- É encorajada a participação ativa dos utilizadores de modo a identificar os verdadeiros requisitos do sistema (diminuindo a probabilidade de interpretações erradas);
- A execução de testes contínuos e iterativos permite uma avaliação objetiva do estado do projeto;
- As inconsistências entre as fases de análise, desenho e implementação são identificadas atempadamente;
- O esforço dos diversos membros da equipa é distribuído ao longo do tempo;
- A equipa pode aprender com experiências anteriores e melhorar a sua atuação;
- Pode ser mostrado, a todos os interessados, o avanço no projeto.

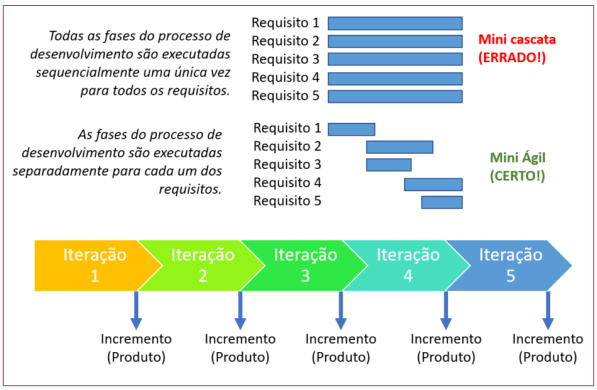
Metodologia Iterativa e incremental

Desvantagens

- Dificuldade de gestão (porque as fases de do ciclo podem estar a ocorrer de forma simultânea);
- O cliente pode entusiasmar-se excessivamente com a primeira versão do sistema e pensar que a mesma já corresponde ao sistema como um todo.

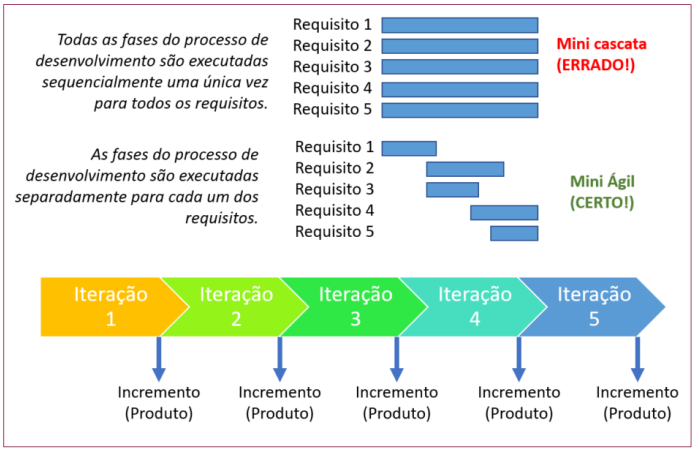
Metodologia ágil (baseada no método iteratividade e incremental)

- As iterações não são divididas em fases;
- É efetuada a divisão dos requisitos em partes menores;
- Seleciona-se um requisito (ou uma parte menor dele), realiza-se a análise, projeto, codificação, integração e teste;
- Entretanto, outro requisito pode ter sido escolhido e o ciclo é executado para esse mesmo requisito. E assim se prossegue até o término da iteração.

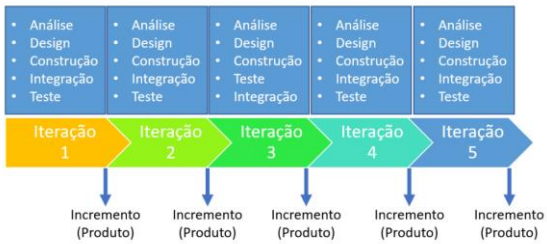


Metodologia ágil

A metodologia ágil permite definir um esboço de um projeto e ir desenvolvendo, em cada iteração, um requisito do sistema/projeto, passando por todas as fases de desenvolvimento.



Metodologia ágil
(baseada na iteratividade)



Vantagens

- Redução de riscos;
- Economia de recursos;
- Entregas dentro do prazo estipulado;
- Resultado final de altíssima qualidade;
- Agilidade e eficiência na execução do projeto;
- Maior alinhamento entre a equipa e os clientes;
- Rápida resolução de possíveis conflitos ou problemas;
- Flexibilidade para propor alternativas para se chegar à melhor solução.

Desvantagens

- Demorará mais a ter uma versão global do projeto.

Exemplos: Kanban, Scrum, Lean

Técnicas de programação

- Uma técnica de programação, independentemente da metodologia adotada, representa uma forma de propor uma solução para o projeto de software a implementar.

Ausência de métodos

- Surgimento de linguagens de baixo nível:
  - Linguagem máquina;
  - Assembly.

Vantagens

- Rapidez;
- Tamanho.

Ausência de métodos

Desvantagens

- Necessidade de muitos conhecimentos técnicos;
- Dificuldade ou impossibilidade de realizar algoritmos complexos;
- Não existência de regras de programação;
- Tempos de desenvolvimento longos.

Métodos algorítmicos

- Linguagens simbólicas (alto nível):
  - Científicas: Fortran, Algol;
  - Comercial: Cobol;

Desvantagens

- Uso excessivo de GOTO;
- Poucas regras de escrita de código;
- Programas potencialmente difíceis de compreender.

Vantagens

- Instruções claras;
- Instruções que não necessitam de muito conhecimento técnico;

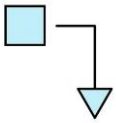
Programação estruturada

- O princípio básico da programação estruturada é que um programa se pode reduzir a apenas três estruturas: sequência, seleção e iteração (ou repetição).

Exemplos de linguagens com programação estruturada (de alto nível): C, Pascal, Cobol

Houve um aumento na complexidade dos programas e introdução de módulos

Sequência

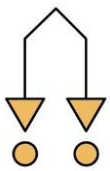


- São implementados os passos de processamento necessários para descrever determinada funcionalidade. Um exemplo básico seria um fluxograma, onde primeiro é executado a Etapa 1 e após a sua conclusão a Etapa 2 é executada, e assim por diante.

Programação estruturada

Seleção

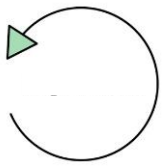
Na seleção, o fluxo a ser percorrido depende de uma escolha.



- A declaração condicional deve ter pelo menos uma condição/escolha verdadeira e cada condição deve ter um ponto de saída no máximo.

Iteração

Na iteração é permito a execução de instruções de forma repetida, onde ao fim de cada execução ou antes da execução dessas instruções, uma condição é (re)avaliada. Apenas quando essa condição for falsa é que a execução programa continua para a instrução seguinte ao bloco de instruções incluídas na estrutura d iteração.



Programação estruturada

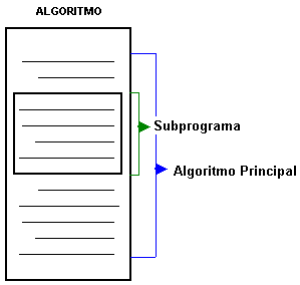
- *Blocos*
- s blocos são usados para permitir que grupos de instruções sejam tratados como se fossem uma instrução.
- Linguagens estruturadas em bloco têm uma sintaxe para estruturas encerradas de alguma forma formal, como...
- Ou recuo de espaço em branco como em Python
- ou entre palavras reservadas BEGIN..END como em Cobol
- uma instrução if entre colchetes {} em C,

```
//test program
#include <iostream>
int main ()
{
std::cout << "Hello World!";
}
```


Programação estruturada

Modularização

- A medida que o sistema vai tomando proporções maiores, é mais viável que o mesmo comece a ser divido em partes menores, onde é possível simplificar uma parte do código deixando a compreensão mais clara e simplificada.
- Essa técnica ficou conhecida como Subprogramação ou Modularização sendo usada através de procedimentos, funções, métodos, rotinas e uma série de outras estruturas.
- Com essa divisão do programa em partes podemos extrair algumas vantagens, como:
  - Cada divisão possui um código mais simplificado;
  - Facilita o entendimento, pois as divisões passam a ser independentes;
  - Códigos menores são mais fáceis de serem modificados;
  - Desenvolvimento do sistema através de uma equipe de programadores;
  - Reutilização de trechos de códigos.



Programação estruturada

Vantagens

- Capacidade de modularização ou criação de subrotinas;
- Propõe uma forma de raciocínio intuitivo, onde há legibilidade e compreensão de cada bloco de código (código organizado);
- Maior facilidade em realizar tarefas complexas;

Desvantagens

- Aumento da complexidade geral;
- não se consegue que uma variável pode ser acedida por apenas alguns subprogramas do programa, levando à utilização excessivo de variáveis globais
- Foca-se em como a tarefa deve ser feita e não em o que deve ser feito;
- O código pode ser confuso;
- Orientada para a resolução de um problema em particular.

Programação orientada a objetos

- Baseada na composição e interação entre diversas unidades, chamadas de 'objetos'
- O intuito da sua criação também foi o de aproximar a lógica de organização de um programa ao manuseamento e relacionamento das coisas do mundo real, daí o nome "objeto" como uma algo genérico, que pode representar qualquer coisa tangível.
- Os objetos da POO (Programação orientada a objetos), podem conter dados na forma de campos, também conhecidos como **atributos** e códigos, e na forma de procedimentos, conhecidos como **métodos.**
- Surge pelo aumento da complexidade e necessidade de estruturas de apoio:
  - Frameworks;
  - Middleware;
  - Componentes;
  - Ambiente

  Fala-se também na abstração: Representação de uma ideia ou **objeto** através das características essenciais do mesmo.

Programação orientada a objetos

Conceitos chave

- Objeto;
- Classe.

Princípios orientadores

- Encapsulamento da informação;
- Herança;
- Polimorfismo;
- Abstração.

Programação orientada a objetos

Vantagens

- Reutilizar código;
- Utilizar o mesmo padrão num projeto;
- Melhor ligação utilizador/desenvolvedor;
- Representação do sistema muito mais próxima da vida real;
- Sistemas com vida útil mais longa;
- Desenvolvimento acelerado;
- Criar aplicações mais complexas;
- Menor custo de manutenção.

Programação orientada a objetos

Desvantagens

- Aprendizagem pode ser complexa;
- Maior uso de memória;
- Maior esforço na planificação;
- Funcionalidades limitadas por interface;
- Dependência de funcionalidades já implementadas em superclasses.

Programação orientada a objetos

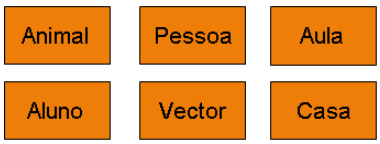
As **Classes** podem representar:

Coisas concretas:
Pessoas, Turma, Carro, Pássaro, Imóvel, Fatura, Livro, Animal

Papéis que coisas concretas assumem:
Aluno, Professor, Piloto

Atividades: Curso, Aula, Acidente

Tipos de dados: Data, Intervalo de Tempo, Vetor



"Uma classe é um modelo para algo que deve ser criado "

Uma classe é a representação de alguma entidade ou coisa que é composta por estados e comportamentos.

Programação orientada a objetos

Objetos

- Representam:
  - Entidades;
  - Conceitos;
  - Estruturas de dados;
- Têm:
  - Estado (atributos);
  - Comportamento (responder a perguntas; alterar o estado dos seus atributos);
  - Identidade.

Programação orientada a objetos

Membros de uma Classe (objetos)

Uma classe é uma estrutura constituída por:

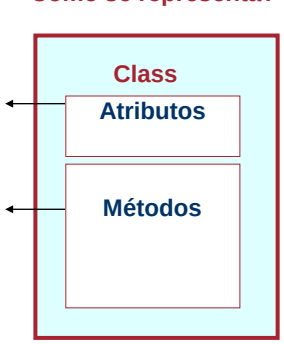
Atributos ou propriedades

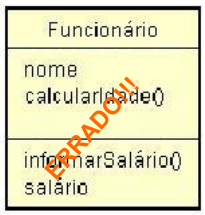
- Um atributo é uma **caraterística** duma classe a assumem um determinado estado;
- Os atributos são uma descrição das propriedades da classe, identificando-a;
- Guardam informação.

Métodos

Operações, ações que caraterizam o comportamento de um objeto, e são o único meio de aceder, manipular e modificar os atributos de um objeto.

Como se representa?





Programação orientada a objetos

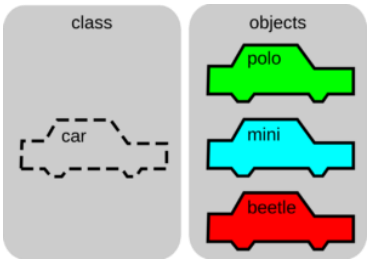
Classes e objetos

OBJETO
Instância de uma classe, é um exemplar da classe;

É criado e manipulado durante a execução do programa;

Tem identidade e valores específicos para os seus atributos.

Tem os seus estados próprios, através dos valores dos seus atributos



Programação orientada a objetos

Características dos atributos

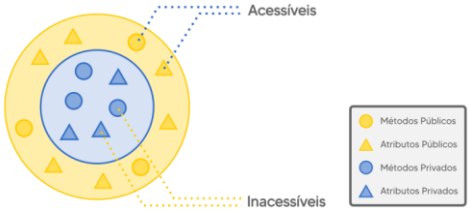
- Domínio dos valores;
- Unidades de medida;
- Valor por defeito;
- Valor inicial;
- Condições de leitura e escrita;
- Condições relativas a outros atributos do objeto ou classe.

Programação orientada a objetos

Programação orientada a objetos

Encapsulamento

- Refere-se ao isolamento entre as partes de um programa e é o processo de ocultação de detalhes de um objeto
- O objetivo do encapsulamento é controlar o tipo de acesso às classes, atributos e métodos o que é conseguido através dos modificadores de acesso*.
- É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.



* definem o tipo de visibilidade que um atributo ou método de um objeto de uma classe

Programação orientada a objetos

Encapsulamento

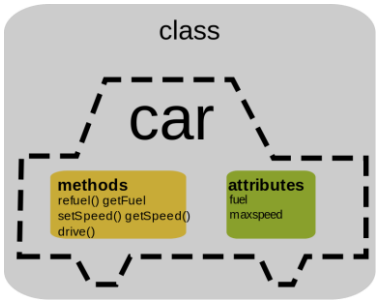
Um carro possui atributos e métodos, ou seja, características e comportamentos. Possui um tanque de gasolina e acelera, por exemplo.

Como sabemos como o nosso carro acelera? É simples: não sabemos.

Na verdade, não importa para o programa como o objeto o faz, só que ele o faça.

Nós só sabemos que para acelerar, devemos pisar no acelerador e de resto o objeto sabe como executar essa ação sem expor como o faz.

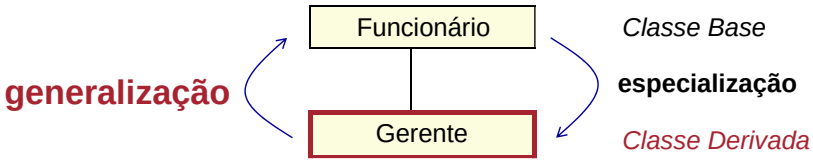
Dizemos que a aceleração do carro está *encapsulada*, pois sabemos o que ele vai fazer ao executarmos esse método, mas não sabemos como – e na verdade não importa ao programa como o objeto o faz, só que ele o faça.



Programação orientada a objetos

Herança

- Definição das relações entre classes e subclasses e superclasses (atributos e operações):
  - Partilha;
  - Acrescento;
  - Redefinição;
- Permite:
  - Reutilizar elementos comuns (generalização);
  - Acrescentar detalhes específicos (especialização).



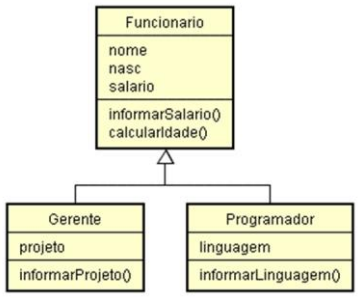
Programação orientada a objetos

Herança

É uma propriedade existente no paradigma orientado a objetos que permite a **reutilização** da estrutura e do comportamento de uma classe ao definir-se novas classes.

HIERARQUIA

GERAL



ESPECÍFICO

Classe Base ou superclasse (caraterísticas comuns)

Classe Derivada ou filha (caraterísticas específicas) Possui os atributos e métodos da classe herdada

- Capacidade de classes mais concretas herdarem comportamentos e atributos, através dos métodos e atributos public/protected, das suas superclasses ou classes base.

Programação orientada a objetos

Polimorfismo

- O termo polimorfismo o é originário do grego e significa "**muitas formas**".
- Aplicado a linguagem orientada por objetos, o polimorfismo é uma técnica de permite implementar código para um objeto em que são criadas várias versões desse objeto para ser utilizado em contextos diferentes, com um modo de funcionamento adequado a cada contexto.
- O polimorfismo **aplica-se apenas aos métodos da classe** e, por defeito, exige a utilização da herança.
- A comunicação é estabelecida apenas na classe mais alta da hierarquia, contudo, como as subclasses estão ligadas por herança, é possível essas classes ”receberem" a comunicação.
- É uma forma de objetos diferentes responderem a uma mesma mensagem

Programação orientada a objetos

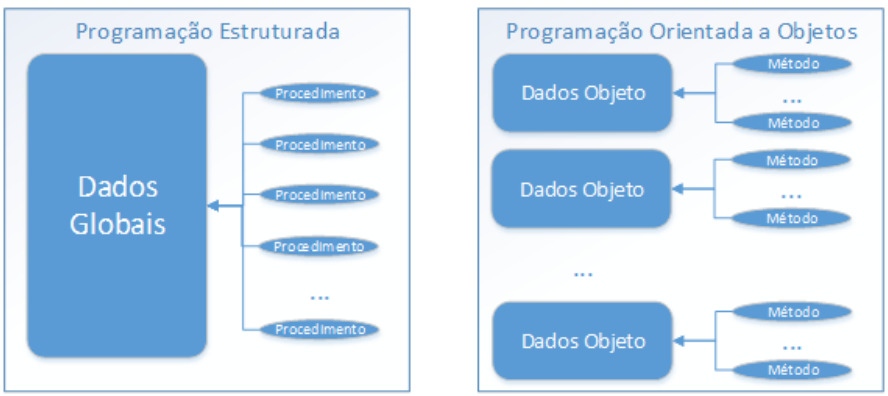
Como definir o comportamento especifico para cada subclasse?

Sobrescrita de método

Cada animal emite sons diferente.
Leão: Rugir
Tigre: Roncar
Cachorro: Latir
Gato: Miar, rosnar
Lobo: Uivar;

Polimorfismo

Programação estruturada versus orientada a objetos



procedimentos (ou funções) que são aplicados globalmente em no programa	métodos que são aplicados aos dados de cada objeto

Essencialmente, os procedimentos e métodos são iguais, sendo diferenciados apenas pelo seu objetivo

Programação orientada a dados

- abordagem de programação em que os dados são organizados da forma que melhor faz sentido para a aplicação aproveitar melhor o hardware, independentemente do código ficar mais organizado ou não;
- O objetivo é a organização dos campos em estruturas de dados para melhor aproveitar a memória, a cache, o transporte e o processamento de dados;
- Em geral diminui a ociosidade da máquina evitando o gargalo de Von Neumann que impede o processador de exercer o seu potencial , devido à forma como os dados estão estruturados
- Esta abordagem popularizou-se em meados dos anos 2000 com a 7.ª geração de consolas e videojogo, onde as aplicações exigem o máximo desempenho. Uma das linguagens que a usa é o C++.
- a tendência de um processador aceder o mesmo conjunto de locais de memória repetidamente, durante um curto período de tempo, tem sido usada para controlar o desempenho dos jogos . Esta abordagem vai de encontro a este objetivo.


```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

(6/32) = ~5.33 loop/cache line
Sqrt + math = ~40 x 5.33 = 213.33 cycles/cache line