Este EU mas Distribuídos

Sobre Java e metodologias de programação

António Rui Borges

Resumo

- *Java*
  - *Nota histórica*
  - *ures*
- *Descrição geral técnicatécnicas o problema em solução*
- *Metodologias de programação*
  - *Programação processual ou imperativa*
    Façanha principal
  - *Programação modular*
  - *Programação orientada a objetos*
  - *Programação simultânea*
  - *Programação distribuída*
- *Leitura sugerida*

Nota histórica - 1

Java, originalmente chamado *Oarvalho*, foi criado em meados da década de 90 por um *Verde* grupo de trabalho do projeto da Sun Microsystems, liderado por James Gosling

- sua estrutura sintática é baseado em batendo linguagens C e C++
- a área alvo inicial da linguagem era a construção de ambientes de controle e comunicação para sistemas embarcados de aparelhos eletrônicos de consumo
  - sua popularidade, no entanto, surge em um contexto diferente, o de *Internet*: primeiro, com *miniaplicativos* (pequenos programas escritos em Java que podem ser executados dentro de um *navegador*) e, mais recentemente, com *serviços web* e programa
- numa perspectiva um pouco mais ampla, Java tende a ser a linguagem de escolha para escrever aplicações distribuídas, uma vez que fornece um ambiente integrado e uniforme para comunicação entre processos através de uma rede de computadores (Java em ação).

Nota histórica - 2

Os objetivos iniciais do Java, como linguagem de programação, incluíam

- *ser totalmente independente da plataforma de hardware subjacente*, isso levou a ser simultaneamente pensado como um *linguagem* e como um *ambiente de execução*: assim os programas podem ser transferidos em tempo de execução e executados em qualquer nó da malha de processamento
- *ser inerentemente robusto*, minimizar erro ors: é, portanto, organizado como um linguagem fortemente semântica onde algumas características do C++, de herança múltipla e de sobrecarga de operadores, por exemplo, foram eliminadas por serem julgadas potenciais fontes de erro; um mecanismo de gerenciamento implícito de memória dinâmica (*coleta de lixo*) também foi introduzida e qualquer tentativa de definição de estruturas de dados fora do escopo doaulaconstrutor foi proibido; nesse sentido, algumas pessoas afirmam que ***Java é C++ bem feito***

  programa

- *para promover a segurança* em ambientes que trocam continuamente informações e compartilham códigos: *ponteiros* foram assim eliminados, tentando evitar que programas não autorizados conseguissem aceder a estruturas de dados residentes na memória.

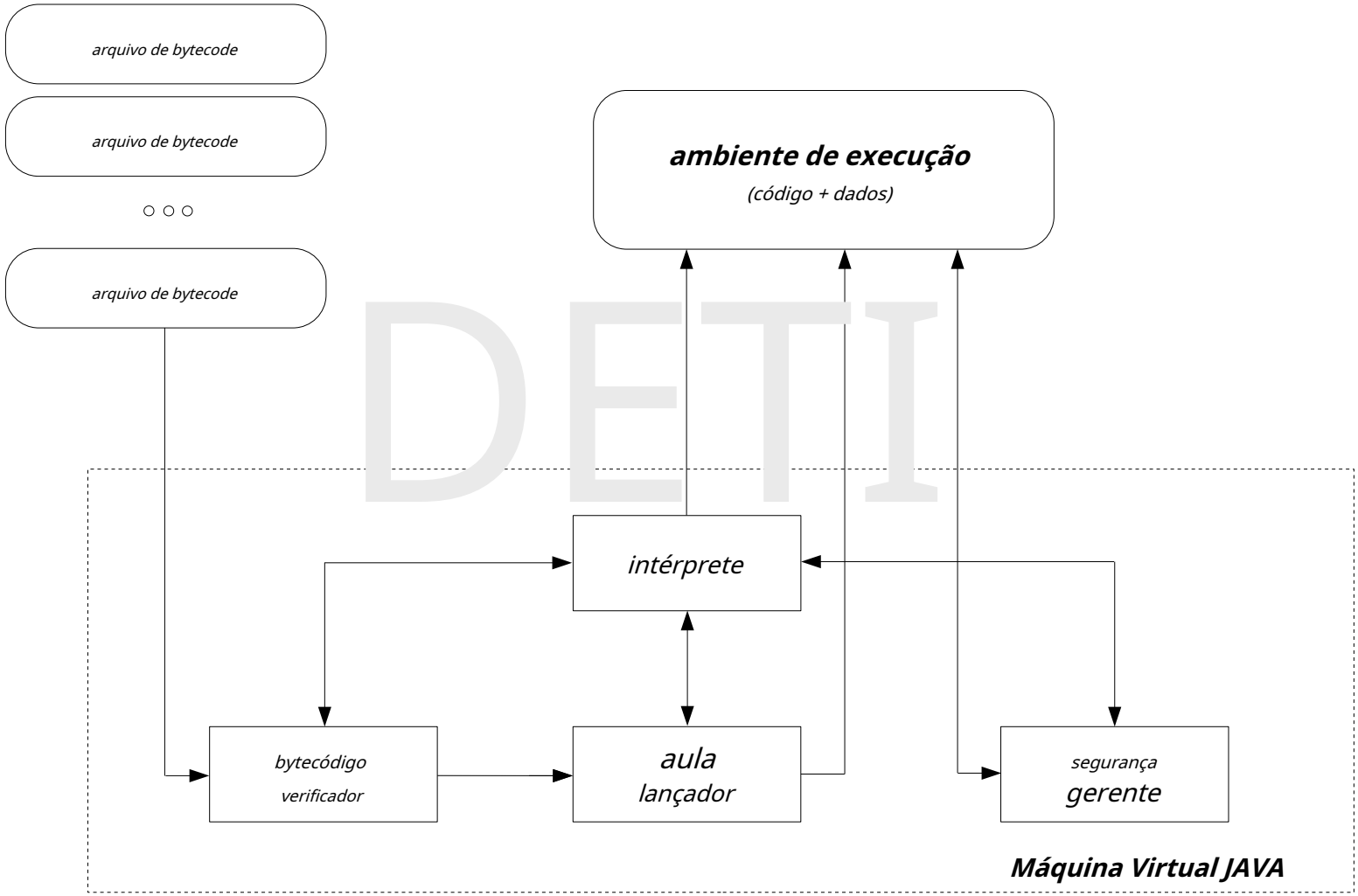
Principais características - 1

- a linguagem segue os princípios do *paradigma orientado a objetos*
  - existe um número mínimo de construções semânticas, permitindo uma descrição compacta e inerentemente segura
  - o mecha de herança nismo é estrito muito linear
  - ointerfaceconstrutor, como uma especificação caso real doaulaconstrutor, foi introduzido para permitir uma separação precisa e clara entre especificação e implementação e para servir como um ponto de conexão entre variáveis de tipos de dados distintos
- fornece construções de simultaneidade
  - há um apoio explícito à construção *multithreading* ambientes
  - *objetos* são implicitamente transformados em *monitores* [do tipo Lampson/Redell] através da sincronização de seus métodos públicos
- suporta programação distribuída, fornecendo bibliotecas que implementam os dois principais paradigmas de comunicação
  - passagem de mensagem: *tomadas*
  - variáveis compartilhadas: *método remoto de invocação* (RMI)

Principais características - 2

- o ambiente de execução Java, chamado *Máquina Virtual JAVA* (JVM), constitui um *intermediário* camada que o aplicativo cações sejam totalmente independentes dente da plataforma de hardware e do sistema operacional onde são executados, implementando um modelo de segurança de três camadas que protege o sistema contra códigos não confiáveis
  - o *verificador de bytecode* analisa o bytecode apresentado para execução e garante que as regras básicas da gramática Java sejam obedecidas
  - o *lançador de classe* entrega os tipos de dados necessários ao interpretador Java
  - o *gerente de segurança* trata dos problemas que potencialmente colocam em risco a segurança do sistema relacionado à aplicação, controlando as condições de acesso do programa em execução ao sistema de arquivos, à rede, aos processos externos e ao sistema de janelas

Principais características - 3



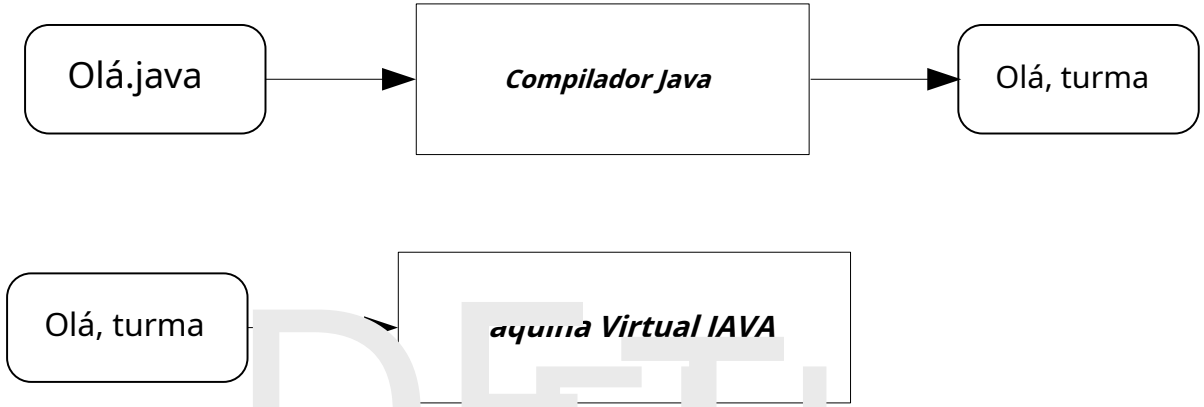
Principais características - 4

- apoia a internacionalização
  - O código ASCII é substituído pelo Unicode na representação interna de caracteres para permitir a consideração de todos os alfabetos
  - a separação das informações de localidade do código executável, juntamente com o armazenamento de elementos de texto fora do código-fonte e seu acesso dinâmico, garantem que haverá uma conformidade das aplicações com o idioma e outras características culturais específicas do usuário final
- inclui ferramentas que facilitam a produção de documentação do programa
  - tendo em conta os *comentários de documentação* inserido nos arquivos fonte para descrever os tipos de dados de referência, suas estruturas internas de dados e seus métodos de acesso, torna-se trivial gerar documentação em *HTML* formato através da aplicação do javadoc ferramenta.

Exemplo trivial - 1

```
/**
 *      Descrição geral:
 *        neste caso, pode ser que o programa imprima a frase
 *          <em>Olá <b>nome da pessoa</b>, como vai você?</em> na tela do monitor.
 *
 *      @autor António Rui Borges
 *      @versão 1.0 - 13/2/2017
 */

aula pública Olá

{
   /**
    *      Programa principal,
    *          isso é implementado pelo método principal e não pelo método de teu tipo de dados.
    *      <p>Qualquer aplicação deve conter um método estático chamado main em seu
    *          iniciar tipo de dados.</p>
    *
    *      @param args nome da pessoa a ser saudada
    */

    vazio estático público principal (String [] argumentos)

    {
        System.out.println ("Olá " + args[0] + ", como vai?");
    }
}
```

Exemplo trivial - 2



[ruib@ruib-laptop1 curiosidades alExemplos]$ *lo java*
[ruib@ruib-laptop1 exemplos triviais]$*java Olá Pedro* *jav*

Olá Pedro, tudo bem?

[ruib@ruib-laptop1 exemplos triviais]$*tudo* total 24

- rw-r--r-- 1 ruib ruib 599 13 de fevereiro 07:42 Olá.class
- rw-r--r-- 1 ruib ruib 654 13 de fevereiro 08:06 Olá.java
- rwxr--r-- 1 ruib ruib 131 13 de fevereiro 06:42 Olá.javadoc
- rw-r--r-- 1 ruib ruib 424 13 de fevereiro 08:07 HelloWorldApp.class
- rw-r--r-- 1 ruib ruib 149 26 de novembro de 2015 HelloWorldApp.html
- rw-r--r-- 1 ruib ruib 374 13 de fevereiro 08:03 HelloWorldApp.java
[ruib@ruib-laptop1 trivialExamples]$

Exemplo trivial - 3

Produção de documentação

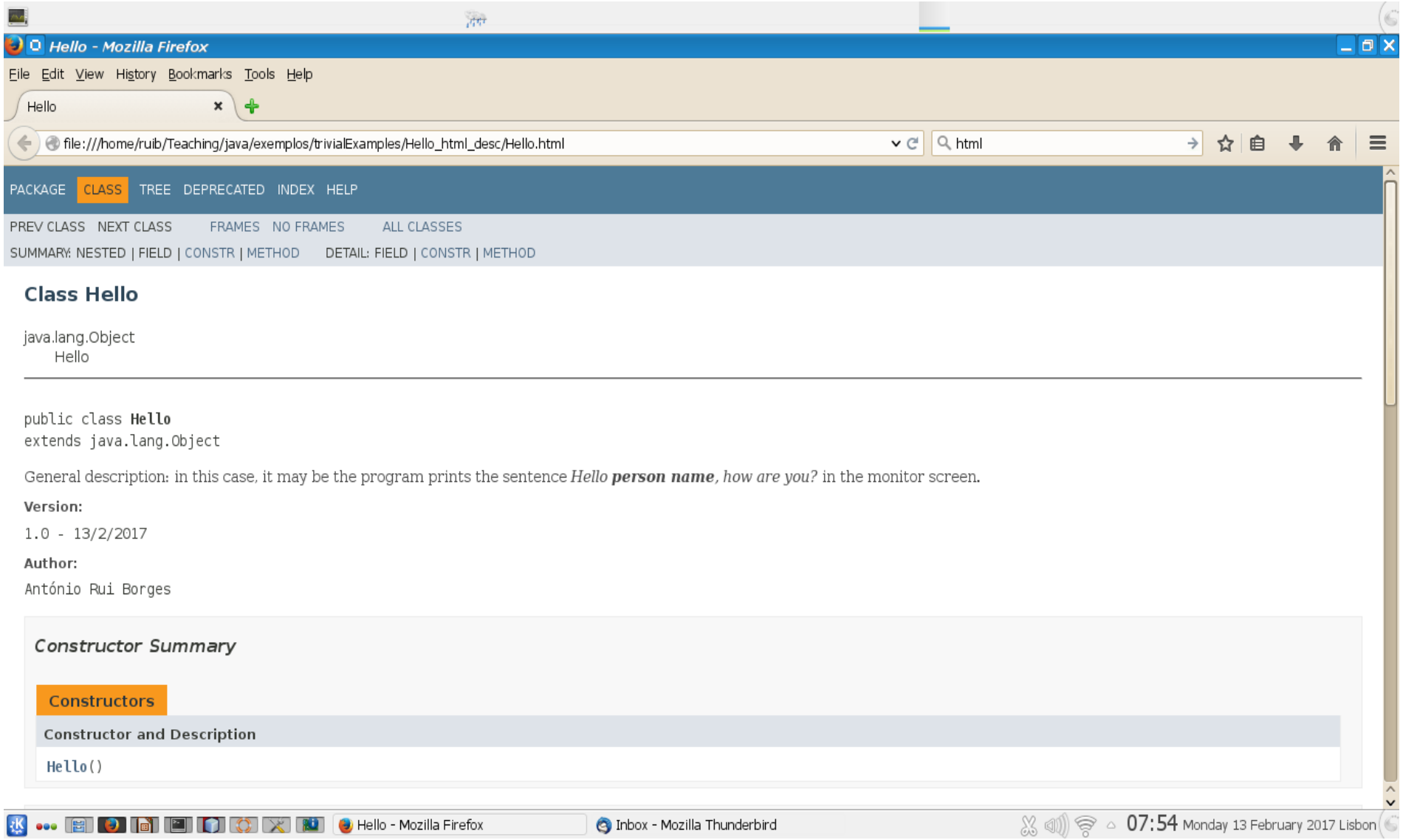
[ruib@ruib-laptop1 exemplos triviais]$*gato Olá.javadoc*
javadoc -d Hello_html_desc -author -version -breakiterator -charset "UTF-8" Hello.java ln -s Hello_html_desc/Hello.html Hello.html

[ruib@ruib-laptop1 exemplos triviais]$*./Olá.javadoc* Carregando arquivo fonte Hello.java... Construindo informacões Javadoc...

Criando diretório de destino: "Hello_html_desc/" Doclet padrao versão 1.8.0_31
Construindo árvore para todos os pacotes e classes... Gerando Hello_html_desc/Hello.html... Gerando Hello_html_desc/pacote-frame.html...
Gerando Hello_html_desc/package-summary.html . . .
Gerando Hello_html_desc/package-tree.html... Gerando Hello_html_desc/constant-values.html... Construindo índice para todos os pacotes e classes... Gerando Hello_html_desc/overview-tree.html... Gerando Hello_html_desc/index-all .html... Gerando Hello_html_desc/deprecated-list.html... Construindo índice para todas as classes...

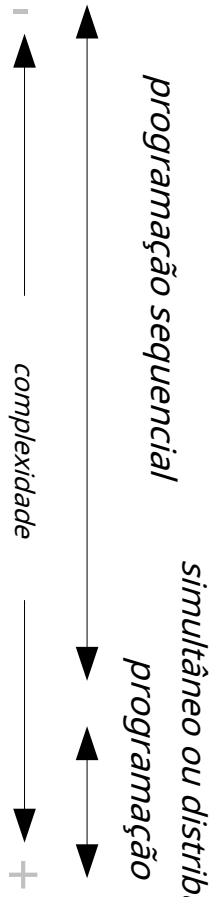
Gerando Hello_html_desc/allclasses-frame.html... Gerando Hello_html_desc/allclasses-noframe.html... Gerando Hello_html_desc/index.html...
Gerando Hello_html_desc/help-doc.html... [ruib@ruib-laptop1 trivialExamples]$

Exemplo trivial - 4



Técnicas de descrição geral de solução de um problema

- *decomposição hierárquica*
  - recorre-se a uma metalinguagem de descrição, de preferência semelhante à linguagem de programação que será utilizada
  - a informação é encapsulada através de
    - a construção de tipos para o dados adequados personagem tiques do problema
    - a definição de novas operações no contexto da linguagem
  - dependências de dados estritas são estabelecidas
- *decomposição em estruturas autônomas interativas*
  - especificação precisa de um mecanismo de interação
  - separação clara entre *interface* e *implementação*
    - abstração e proteção de dados
    - virtualização de operações de acesso
- *decomposição em entidades autônomas interativas*
  - especificação precisa de um modelo de comunicação
  - definição e implementação de mecanismos de sincronização.

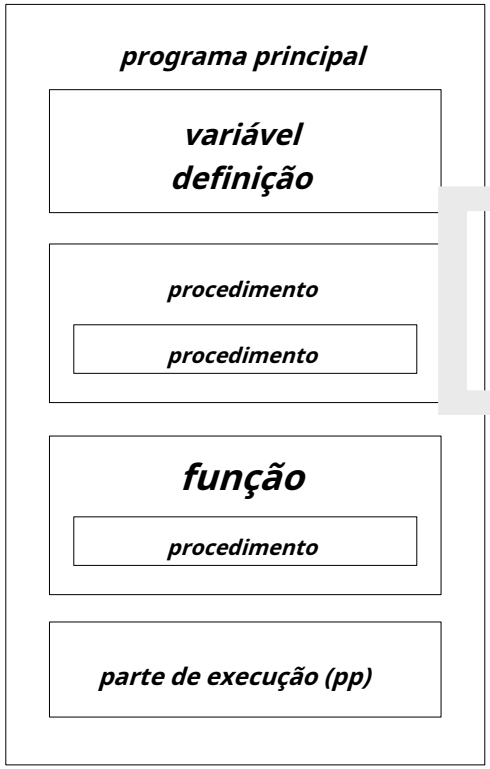


Metodologias de programação - 1

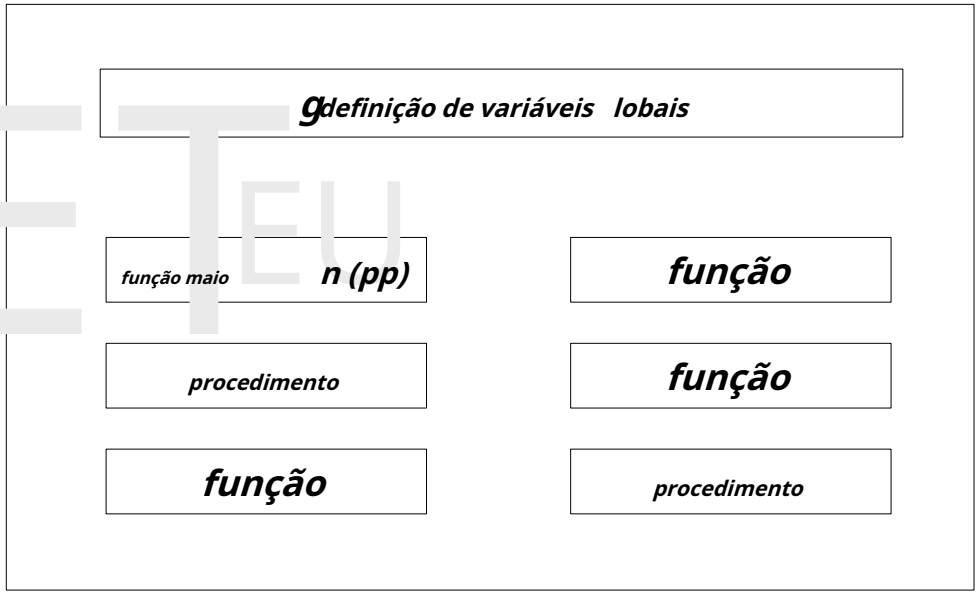
Programação processual ou imperativa

- a ênfase é colocada em uma implementação mais ou menos precisa da decomposição hierárquica da solução
- complexidade é controlada pelo uso intensivo de
  - *funções* e *procedimentos*, como um meio de descrever operações em diferentes níveis de abstração
  - *construção de tipo de dados* ou *diretores*, também chamados de associou
- organizar informações mais informadas forma adequada às características do problema
- espaço variável é *concentrado*
  - todos os dados relevantes para a solução do problema são definidos no *programa principal* nível, ou é global; variáveis locais para as funções e procedimentos restantes requerem apenas armazenamento temporário (elas só existem quando as funções ou os procedimentos são invocados)
  - o acesso das diferentes operações aos dados é concebido em estrita obediência ao princípio "*o que eles precisam saber*", usando um modelo de comunicação baseado na transferência de parâmetros

Programação processual ou imperativa



organização hierárquica de um arquivo fonte – Pascal como



organização horizontal de um arquivo fonte – C linguagem como (estrutura do mar de funções)

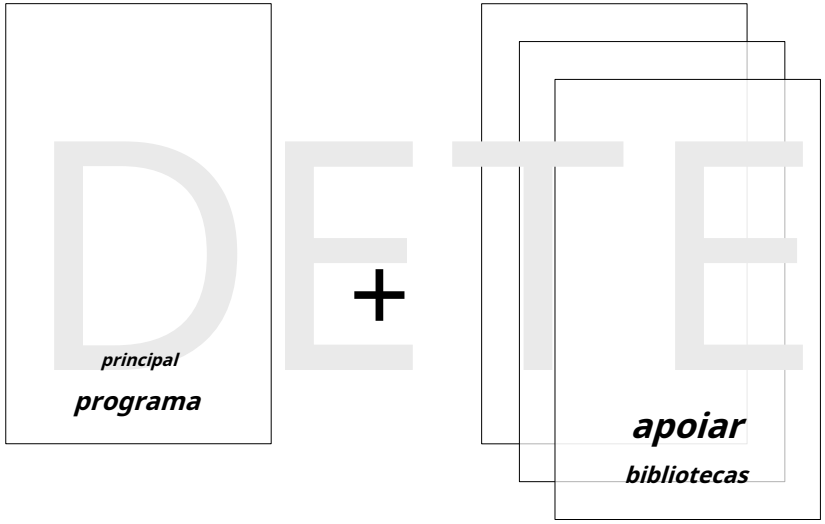
Metodologias de programação - 3

Programação processual ou imperativa

- uma questão que imediatamente vem à mente é *reutilização de código*
- diferentes operações, implementadas por funções e procedimentos no contexto de uma aplicação específica, são frequentemente úteis em outras aplicações
- sua inclusão automática no código de novos aplicativos pode ser feita dividindo o arquivo fonte em vários arquivos, cada um direcionado para compilação separada e reunidos posteriormente em um único arquivo executável na fase de vinculação
- assim, o código de um aplicativo específico fica espalhado por
  - um arquivo fonte que contém o programa principal e, eventualmente, outro código privado
  - múltiplos arquivos fonte que contêm funcionalidades bem definidas, implementadas por funções e procedimentos que foram escritos de forma independente e que serão agora colocados em funcionamento no contexto do presente pedido, o chamado *bibliotecas de suporte*

Metodologias de programação - 4

Programação processual ou imperativa



- torna-se necessário neste tipo de organização inserir nos arquivos fonte por partes, sempre que seu código se referir a operações externas, o nome do arquivo fonte onde são declarados (*arquivo de interface*) para garantir consistência em tempo de compilação.

Metodologias de programação - 5

Programação modular

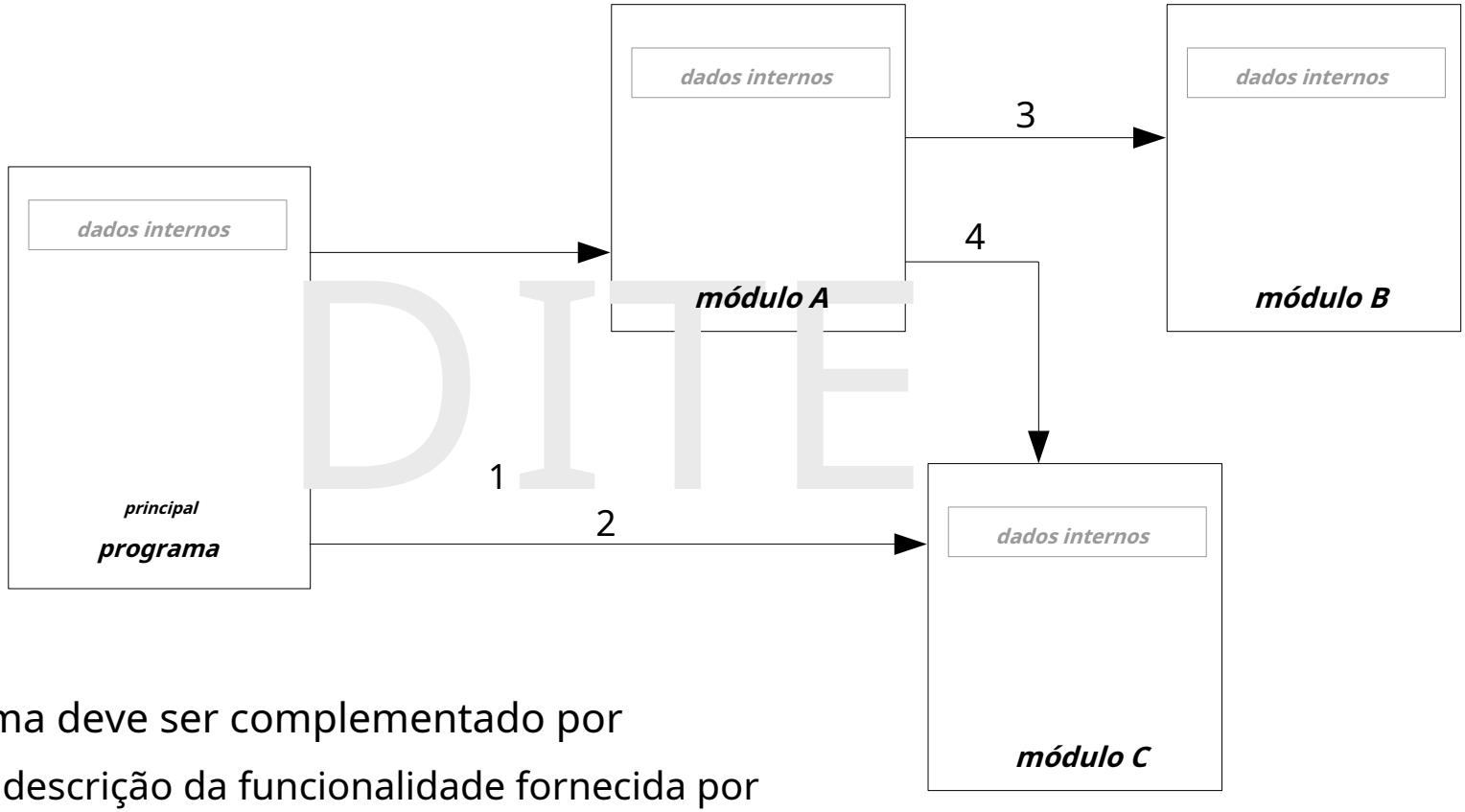
- à medida que a complexidade da descrição da solução aumenta, o gerenciamento centralizado do espaço de variáveis se torna agressivo e muito mais difícil
  - ênfase passa da concepção da operação para o desenvolvimento de uma organização de dados mais refinada
- o espaço de variáveis torna-se de fato *distribuído* e o conceito de *módulo* surge como um meio mais conveniente para lidar com sua gestão
  - a *módulo* é, por definição, uma estrutura autônoma, descrita em um arquivo fonte separado, que encapsula uma funcionalidade bem definida possuindo em geral um espaço proprietário de variáveis e um conjunto de operações para manipulá-la
  - o espaço de variáveis geralmente é *interno*, o que significa que o acesso a ele só pode ser feito pelas operações especificadas, as chamadas *acessar primitivas*
  - deve-se notar que, quando o espaço proprietário de variáveis não tem significado externo, um *módulo* torna-se o que antes era chamado de *biblioteca de suporte*

Metodologias de programação - 6

Programação modular

- a aplicação de um fu funcional u abordagem t solução n decomposição gera o especificação de um conjunto de estruturas autônomas de interação e é possível lidar com problemas muito mais complexos através da criação de uma divisão clara de tarefas e da promoção do trabalho cooperativo
  - a descrição é agora feita especificando precisamente quais são as estruturas interagentes em jogo e estabelecendo para cada uma delas qual é o seu papel na interação e como o acesso a ela deve ser feito
  - a separação óbvia entre especificação de acesso, o *interface*, e a implementação permite sua utilização no projeto de outros módulos, mesmo antes do módulo original estar totalmente concluído e validado

Programação modular



- o diagrama deve ser complementado por
  - uma descrição da funcionalidade fornecida por cada um dos módulos
  - uma listagem das operações envolvidas na interação

Metodologias de programação - 8

Programação modular

- a aplicação consistente de um *dividir e conquistar* estratégia, que está subjacente a qualquer metodologia de descrição e, particularmente, quando é realizada uma decomposição funcional da solução, permite o desenvolvimento de *robusto* código, incorporado por
  - *projeto para teste*: como so**odule é um** estrutura nossa re, sua funcionalidade pode e deve ser validado de acordo com a especificação previamente estabelecida antes de sua inclusão na aplicação que se pretende fazer parte
  - *programação por contrato*: o programador, a quem foi atribuída a tarefa da sua implementação, deverá conceber meios que garantam que apenas a funcionalidade descrita seja disponibilizada em *tempo de execução*
    - todas as primitivas de acesso devem retornar *status* informações sobre a operação: *sucesso* ou *ocorreu um erro*, com uma indicação específica do erro neste último caso
    - a consistência da estrutura de dados interna deve ser garantida anterior para a execução da primeira operação (*pré-invariante*)
    - sempre que uma operação é chamada, o valor de cada variável na lista de parâmetros de entrada deve ser verificado para afirmar se está dentro da faixa permitida, antes da execução da operação
    - a consistência da estrutura de dados interna deve ser garantida depois execução da operação (*pós-invariante*).

autônomo

Metodologias de programação - 9

Programação orientada a objetos

- uma implementação modular de um funcional decompos pode tornar-se bastante inflexível xível em termos de *reutilização de código* sempre que a nova aplicação exigir
  - a instanciação múltipla de um módulo
  - a introdução de pequenas alterações nele
- o programador tenta inevitavelmente lidar com a situação ajustando os módulos pré-existentes às necessidades actuais, conduzindo a uma pulverização progressiva do seu repertório de funcionalidades implementadas e à consequente perda de eficiência na gestão de software de suporte ao desenvolvimento de futuras aplicações
- uma possível solução para este enigma é uma abordagem de maior abstração

Metodologias de programação - 10

Programação orientada a objetos

- o primeiro problema pode ser resolvido b *missão* para O conceito de *tipo de dados*
- na verdade, se fosse possível associar a cada instanciação de módulo uma variável diferente, a ambiguidade seria removida e múltiplas instanciações tornar-se-iam triviais
- nesse sentido, a noção de *tipo de dados* como um conjunto de regras que permitem o armazenamento na memória de valores com determinadas propriedades, é agora alargado para contemplar não só o armazenamento de um agregado de valores, mas também a identificação das operações que podem ser executadas sobre eles sim *extenso*
- embora de forma inadequada, esses tipos de dados são às vezes chamados *tipos de dados abstratos*; *tipos de dados de referência*, ou *tipos de dados definidos pelo usuário*, são nomes melhores que também são comumente usados

Metodologias de programação - 11

Programação orientada a objetos

- linguagens de programação que os suportam, possuem um construtor, cujo nome tradicional é aula, qual permite o associados de uma definição de módulo para um identificador
- a partir deste ponto, torna-se possível declarar variáveis deste tipo
- um detalhe importante é que a declaração pura de variáveis deste tipo não aloca espaço de memória para seu armazenamento, apenas aloca espaço de memória para um *ponteiro*
- a alocação de espaço para o valor em si é feita apenas pelo *tipo instanciação* em *tempo de execução*; então, é alocado espaço na zona de definição dinâmica do espaço de endereço do processo e ocorre a consequente inicialização da estrutura de dados interna; além disso, sempre que a variável não for mais necessária, o espaço de armazenamento na memória dinâmica deverá ser liberado
- os valores instanciados são chamados *objetos*, daí o nome do paradigma

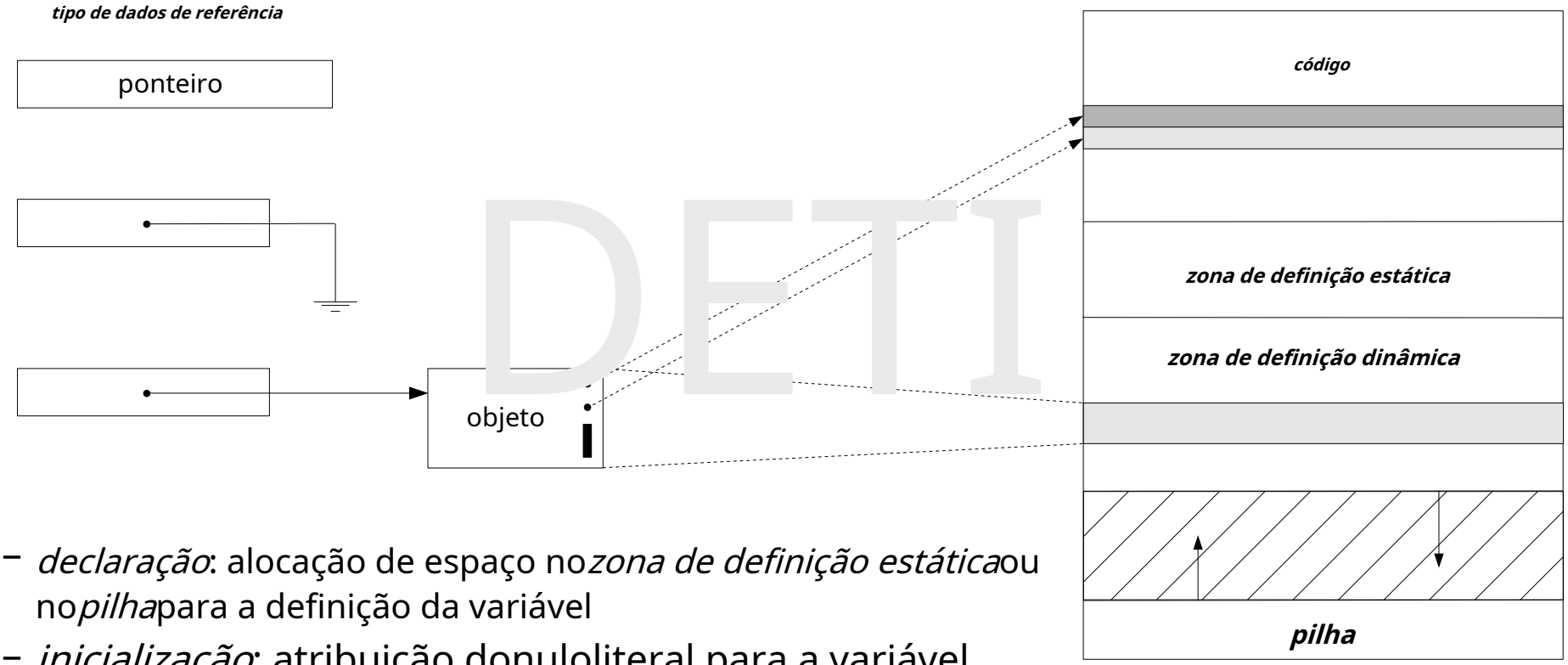
Metodologias de programação - 12

Programação orientada a objetos

- em *Orientado a Objeto* terminologia, chama-se *Campos* às variáveis da estrutura de dados interna do tipo de dados de referência, e *métodos* às primitivas de acesso; um *objeto* é então determinado pelo seu *estado* (conjunto de valores atualmente armazenados nos diferentes campos) e por sua *comportamento* (conjunto de métodos de acesso fornecidos)
- linguagens de programação que suportam o paradigma, geralmente permitem *abstração de dados* na especificação da estrutura de dados interna, dando origem a tipos de dados genéricos com uma gama de aplicações mais ampla
  - em vez de criar um tipo de dados qual eu sou suplementos uma pilha para armazenamento de personagens, para Por exemplo, pode-se criar um tipo de dados que implemente uma pilha para o armazenamento de qualquer tipo de valores
- além disso, ainda se pode garantir que um único tipo de valores seja armazenado lá em *tempo de execução* usando *parametrização*
  - tomando o exemplo anterior, pode-se criar um tipo de dados que implemente uma pilha para o armazenamento de valores de um tipo de dados não especificado Te adiar para a declaração e posterior instanciação de variáveis deste tipo a informação sobre qual tipo T realmente significa

Metodologias de programação - 13

Programação orientada a objetos



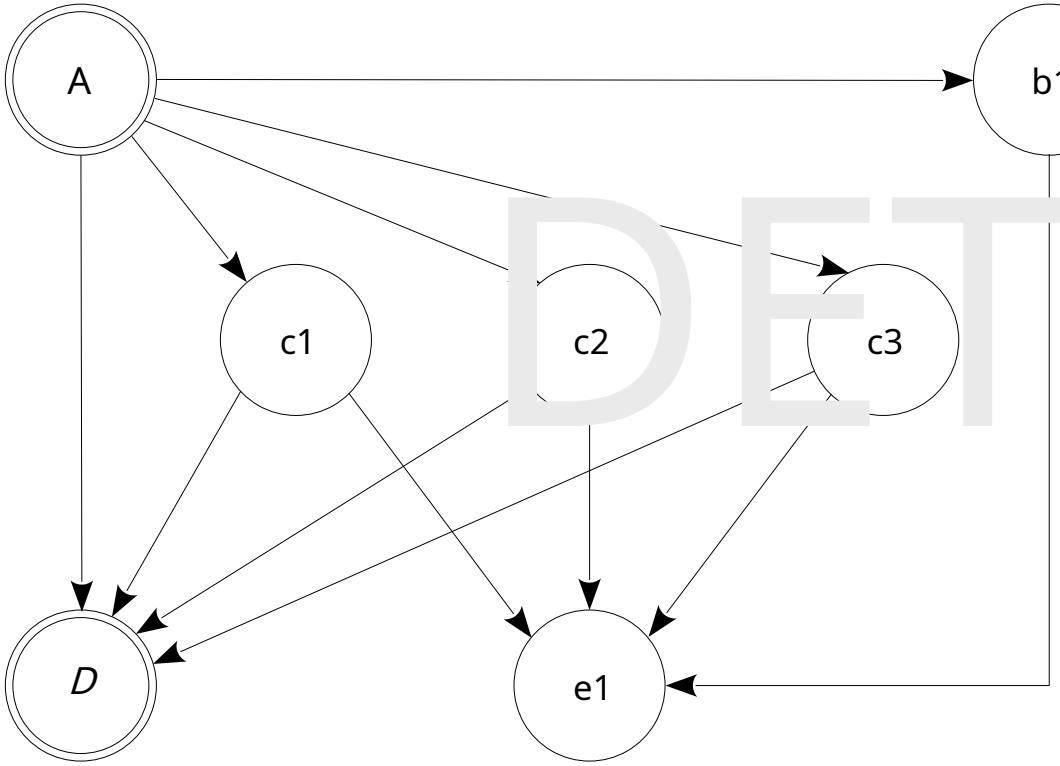
- *declaração*: alocação de espaço no *zona de definição estática* ou no *pilha* para a definição da variável
- *inicialização*: atribuição donuloliteral para a variável
- *instanciação*: alocação de espaço no *zona de definição dinâmica* para *objeto* criação (em *tempo de execução*)

Metodologias de programação - 14

Programação orientada a objetos

- um programa organizado de acordo com o paradigma orientado a objetos consiste em vários arquivos de origem, cada um definindo um tipo de dados específico
- como acontece em modula programa r interação n descrição é retratada por um diagrama que inclui o *não instanciado* e a *instanciado* tipos de dados, os primeiros nomeados pelos identificadores de tipo e os segundos pelos identificadores das variáveis associadas; a forma como interagem é representada por um gráfico orientado que expressa a geometria do acesso; o diagrama deve ser complementado com uma descrição da funcionalidade que cada tipo de dado fornece e com a listagem das operações ing. o
- como uma regra, *não instanciado* tipos de dados representam o *programa principal* e grupos de operações genéricas organizadas em *bibliotecas*; todos os tipos de dados restantes são, em princípio *instanciado*

Programação orientada a objetos



A – tipo de dados que contém o programa principal
b1–instanciação do tipo de dadosB *c1*–instanciação do tipo de dadosC *c2*–instanciação do tipo de dadosC *c3*–instanciação do tipo de dadosC
D–tipo de dados que contém métodos genéricos
e1–instanciação do tipo de dadosE

- –instanciação, acesso a um campo ou invocação de método

mar de tipos de dados não instanciados e instanciados (classes e objetos)

Metodologias de programação - 16

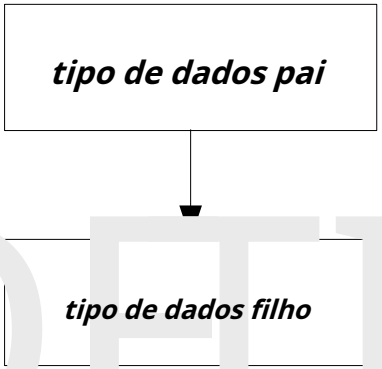
Programação orientada a objetos

- o segundo problema pode ser resolvido pela aplicação do conceito de *reuso* para a construção de tipos de dados
- na verdade, se fosse possível dispor de uma linguagem capaz de definir novos tipos de dados a partir de tipos de dados pré-existentes, pode-se conseguir sua diferenciação para torná-los adequados a novas situações com um mínimo de esforço
- esse mecanismo, chamado *herança*, permite estender de forma hierárquica para o novo tipo de dados, *tipo de dados filho* ou *subtipo*, as propriedades do tipo de dados no qual a definição se baseia, *tipo de dados pai* ou *supertipo*
- especificamente, isso significa que
  - o *protegido* campos da estrutura de dados interna
  - o *público* métodos

  do *base* tipo de dados (tipo de dados pai) são diretamente acessíveis e, no último caso, modificáveis no *derivado* tipo de dados (tipo de dados filho)

Metodologias de programação - 17

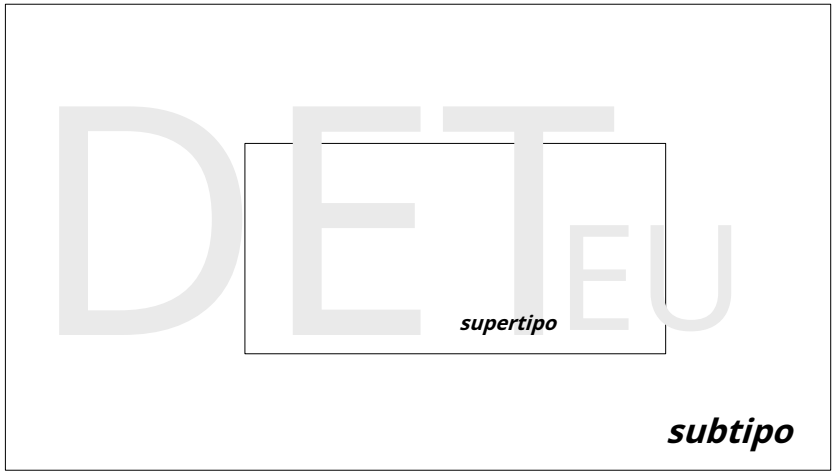
Programação orientada a objetos



- o mecanismo de criação de novos tipos de dados, chamados *derivação* mecanismo, permite
  - introduzir novos campos na estrutura de dados interna
  - para introduzir novos métodos públicos
  - para *sobrepor* métodos: redefinição de métodos públicos [implementados anteriormente] do tipo de dados base
  - para *implementar virtual* métodos: definição de métodos públicos cuja interface está no tipo de dados base

Metodologias de programação - 18

Programação orientada a objetos



- Em termos de compatibilidade, o subtipo é *compatível* com o supertipo do qual é derivado, mas o inverso nem sempre é verdadeiro

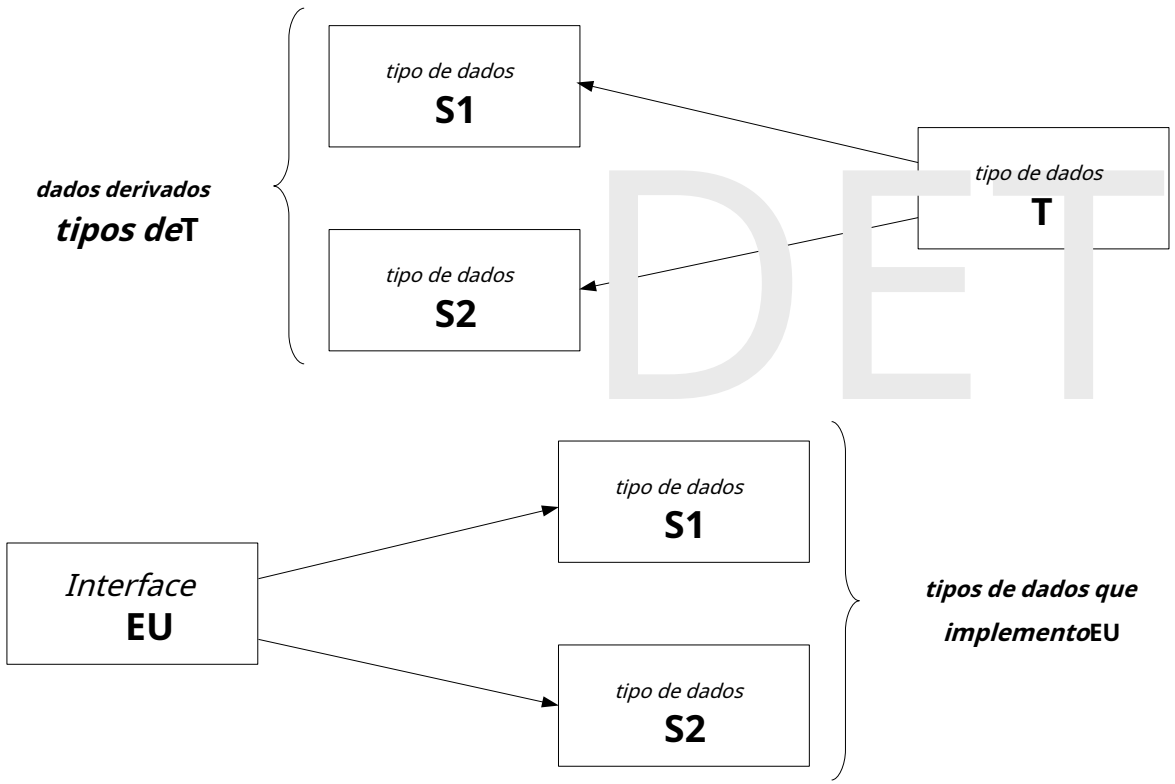
Metodologias de programação - 19

Programação orientada a objetos

- a relação de compatibilidade pode ser usada para estabelecer um princípio muito poderoso do paradigma orientado a objetos, *polimorfismo*, que consiste na possibilidade de utilizar a mesma variável para referenciar objetos de tipos de dados diferentes, derivados de o mesmo tipo de base
- deixar T ser um tipo de dados de referência, S1 e S2 dois tipos de dados distintos derivados do tipo de dados T e t, s1 e s2 variáveis de cada um destes tipos onde foram atribuídos objetos dos respectivos tipos de dados aos dois últimos; considere agora que se atribui qualquer uma dessas variáveis à variável t (sempre uma operação válida porque s1 e s2 são compatíveis com t, embora não sejam mutuamente compatíveis) e invoca-se um método definido em T
- a associação do método adequado ao objeto referenciado é automaticamente selecionada e realizada
  - na fase de compilação, *associação estática*, quando o método acima referido não foi modificado na definição de nenhum dos tipos de dados derivados em jogo
  - em *tempo de execução*, *associação dinâmica*, de outra forma.

Metodologias de programação - 20

Programação orientada a objetos




```
Tt;
eu eu;
S1s1 =novoS1 (...); S2 s2 =
novoS2 (...);

se(doença)
      t = s1;
      outrot = s2;
      . . .
tm (...);          ← invocação de método
                       euna variáveli

      . . .

se(doença)
      eu = s1;
      outroeu = s2;
      . . .
eu sou (...);      ← invocação de método
                       euna variáveli
```

Metodologias de programação - 21

Programação simultânea

- a decomposição funcional das soluções de problemas complexos leva rapidamente à necessidade de conceber um grupo de atividades que pode ativo sendo um grupo de acontecer de forma mais ou menos de forma autônoma e cooperar para alcançar um objetivo comum
- manter um único thread de execução requer a integração no código do usuário de um *Agendador* que alterna entre as diferentes atividades
- esta abordagem, sendo muito complexa e exigente, também é totalmente inútil, uma vez que os sistemas operacionais atuais permitem a multiprogramação, proporcionando facilidades para comunicação e sincronização entre processos.
- além disso, com a popularização *multicore* processadores, sistemas operacionais implementam *fio* gestão no *núcleo* nível, o que significa uma aceleração na execução de aplicações simultâneas

Metodologias de programação - 22

Programação simultânea

- duas abordagens diferentes para design de solução estão em uso
  - *abordagem orientada a eventos*: os processos que implementam as atividades normalmente ficam bloqueados, aguardando que ocorra um evento qual trigonometria gera seu execução; trata-se aqui de processos mais ou menos independentes, geralmente apenas um deles ativo por vez; a comunicação do processo ocorre por meio da escrita e leitura de variáveis compartilhadas mantidas centralmente; *gerenciadores de espaço de visualização*, que interagem com o usuário através do mouse e do teclado, são talvez o exemplo mais comum dessa abordagem
  - *abordagem ponto a ponto*: os processos que implementam as atividades cooperam entre si de forma mais ou menos específica; cada um é pensado como executando um ciclo de vida que consiste em operações independentes e interativas, as últimas produzem ou coletam informações, bloqueiam o processo até que certas condições sejam alcançadas ou despertam outros processos quando certas condições são atendidas.
- seja como for, uma questão sempre presente é que, ao contrário do que acontece na programação sequencial, não há garantia de reprodutividade das operações, portanto, *depuração* é muito mais sensível e muito menos eficaz aqui

Metodologias de programação - 23

Programação simultânea

- existem dois modelos básicos para compartilhamento de informações e comunicação
  - *variáveis compartilhadas*: isto é o comunicador eu entro *multi rosqueado* ambientes onde o os processos intervenientes escrevem e leem valores armazenados em uma estrutura de dados definida centralmente; prevenir *condições de corrida* que possa gerar inconsistência de informações, o código de acesso à região compartilhada (*região crítica*) tem que funcionar em exclusão mútua; há também a necessidade de ter meios para sincronização de processos
  - *passagem de mensagem*: é um modelo de aplicação universal, pois não exige o compartilhamento dos processos endereçando espaço; a comunicação é realizada pela troca de mensagens entre pares de processos (*unicast*), entre um processo e todos os outros (*transmissão*), ou entre um processo e todos os outros pertencentes ao mesmo grupo (*multitransmissão*); assume-se aqui que está disponível uma infraestrutura de canais de comunicação, interligando todos os processos e que é gerida por uma entidade não pertencente à aplicação, garantindo um acesso mútuo exclusivo aos canais e disponibilizando mecanismos de sincronização

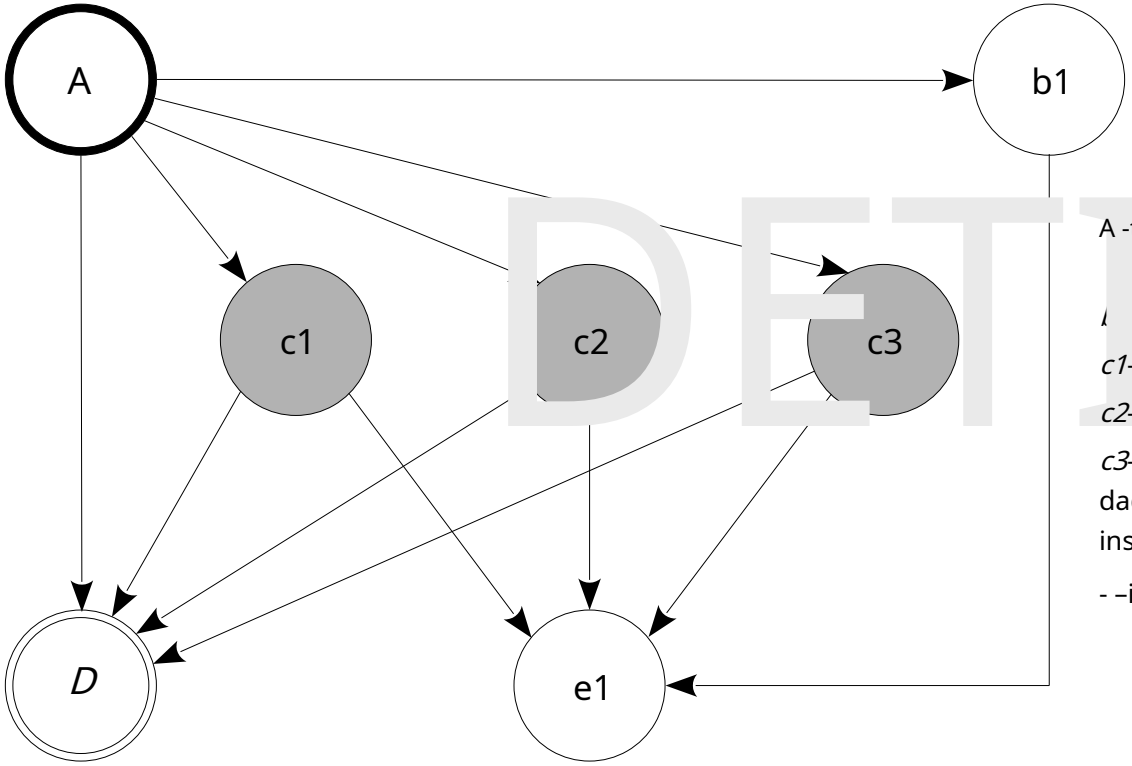
Metodologias de programação - 24

Programação simultânea

- um programa organizou um de acordo com metodologia está em conformidade com os princípios programação modular, ou programação orientada a objetos, para descrever a interação e, portanto, também consiste em vários arquivos de origem
- o que há de novo aqui é que, por ter múltiplos threads de execução, é necessário distinguir claramente entre *ativo* e *passiva* entidades
- *ativo* entidades representam os diferentes processos intervenientes, enquanto *passiva* entidades resultam da explicitação dos diferentes tipos de funcionalidade presentes.

este eu

Programação simultânea



A -tipo de dados que contém o programa principal (entidade ativa)

b1–instanciação do tipo de dadosB

c1–instanciação do tipo de dadosC (entidade ativa)

c2–instanciação do tipo de dadosC (entidade ativa)

c3–instanciação do tipo de dadosC *D*–tipo de (entidade ativa)
dados que contém métodos genéricos *e1*–
instanciação do tipo de dadosE

- –instanciação, acesso a um campo ou invocação de método

mar de tipos de dados não instanciados e instanciados e entidades ativas e passivas

Metodologias de programação - 26

Programação distribuída

- existem duas áreas principais: *programação simultânea* para *programação distribuída*
  - *paralelização*: para aproveitar vários processadores e outros *hardware* componentes de um sistema de computador paralelo para obter uma execução mais rápida e eficiente de uma aplicação
  - *disponibilizar um serviço*: fornecer uma funcionalidade bem definida para um grupo ampliado de aplicações de forma consistente, confiável e segura
- a mudança de metodologia implica, portanto, que se encontre de alguma forma uma forma de mapear em sistemas informáticos distintos os diferentes processos e centros de funcionalidade uma solução concorrente foi anteriormente dividida

Metodologias de programação - 27

Programação distribuída

- o mapeamento não pode ser feito em mático wa sim, no entanto; Existem vários questões no conceito de *plataforma de processamento paralelo* que devem ser cuidadosamente avaliados para que a migração seja possível
- alguns deles são
  - eventuais heterogeneidades entre os nós da plataforma de processamento
  - não existe um relógio global para permitir a ordem cronológica dos eventos
  - alguns nós da plataforma de processamento e partes da infra-estrutura de comunicação podem falhar a qualquer momento.

um automóvel

Leitura sugerida

- *On-line* documento de suporte entrada para programa de ambiente em desenvolvimento por Oracle (Java Platform Standard Edition 8)

r Java pr