Orientação por Objectos – conceitos

Paulo Trigo Silva

Cinco atributos de um sistema complexo

- ... em *Object Oriented Design*, por *Grady Booch*
- "*Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached*"
  - ◊ uma estrutura hierárquica possível de decompor permite,
    - · entender, descrever e visualizar tais sistemas e seus constituintes ...
- "*The choice of what components in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system*"
  - ◊ o que é primitivo para um observador,
    - · pode estar a um nível de abstracção muito alto para outro ...

Paulo Trigo Silva

Cinco atributos de um sistema complexo (cont.)

- "*Intracomponent linkages are generally stronger than intercomponent linkages. This fact has the effect of separating the high-frequency dynamics of the components – involving the internal structure of the components – from the low-frequency dynamics – involving interaction among components*"
  - ◊ a diferença entre ligações "intra" e "inter" das peças do sistema,
    - · favorece a abordagem isolada a cada uma dessas peças ...
- "*Hierarchic systems are usually composed of only a few different kinds of subsystems in various combinations and arrangements*"
  - ◊ tal como as células das plantas e animais, também por vezes,
    - · os subsistemas têm características comuns a diferentes domínios ...

Paulo Trigo Silva

Cinco atributos de um sistema complexo (cont. 1)

- "*A complex system that works is invariably found to have evolved from a simple system that worked ... A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working simple system*"
  - ◊ à medida que os sistemas evoluem, os objectos considerados complexos, tornam-se os objectos primitivos,
    - com base nos quais, sistemas mais complexos são construídos ...
- ... e mais uma ideia – sobre a construção de sistemas complexos,
  - ◊ "*the conception of a design for a new structure can involve as much a leap of the imagination and as much a synthesis of experience and knowledge as any artist is required to bring to his canvas or paper. And once that design is articulated by the engineer as artist, it must be analyzed by the engineer in as rigorous an application of the scientific method as any scientist must make*"
    - o desafio que se coloca ao engenheiro é particularmente grande quando ele tem que construir um sistema inteiramente novo ...

... até à construção de sistemas complexos de *software*

- Não existem "soluções mágicas" que, com sucesso, conduzam
  - ◊ o engenheiro de *software* à implementação de um sistema complexo
- Qual a melhor forma de decompor um sistema complexo ?
  - ◊ que é outra forma da questão – Qual é o melhor método de análise ?
    - · ... e para a qual não existe uma resposta definitiva ...
- Existem diversas abordagens – uma bastante promissora consiste
  - ◊ em focar a abordagem nas "coisas" do domínio do problema ...
    - · ... essência do que se designa por Orientação por Objectos !
- A Orientação por Objectos defende
  - ◊ a decomposição do sistema em constituintes que são objectos,
  - ◊ e onde a construção de modelos contempla os cinco atributos de um sistema complexo ...

Paulo Trigo Silva

... estilos de programação

- "... *programming style is a way of organizing programs on the basis of some conceptual model of programming and an appropriate language to make programs written in the style clear*", em *Object Oriented Design*, por *Grady Booch*

- O autor sugere 5 estilos principais de programação
  - ◊ Orientado ao Procedimento / Algoritmos
  - ◊ Orientado ao Objecto / Classes e Objectos
  - ◊ Orientado à Lógica / Objectivos, expressos num cálculo de predicados
  - ◊ Orientado à Regra / Regras if-then
  - ◊ Orientado às Restrições / Relações Invariantes

- Não existe um estilo melhor para todos os géneros de aplicações
  - ◊ orientado à regra será melhor para lidar com uma base de conhecimento
  - ◊ orientado ao objecto tem-se revelado adequado para uma grande leque de aplicações – em especial para lidar com sistemas complexos ...

Modelo de Objectos

- Modelo de Objectos é uma designação colectiva,
  - ◊ dos princípios em que assenta a Orientação por Objectos
- Os 7 princípios são,
  - ◊ Abstracção
  - ◊ Encapsulamento
  - ◊ Modularidade
  - ◊ Hierarquia
  - ◊ Tipificação
  - ◊ Concorrência
  - ◊ Persistência

4 essenciais – um modelo que não contemple um destes aspectos não se pode considerar Orientado por Objectos

3 menores – cada um destes é útil mas não essencial

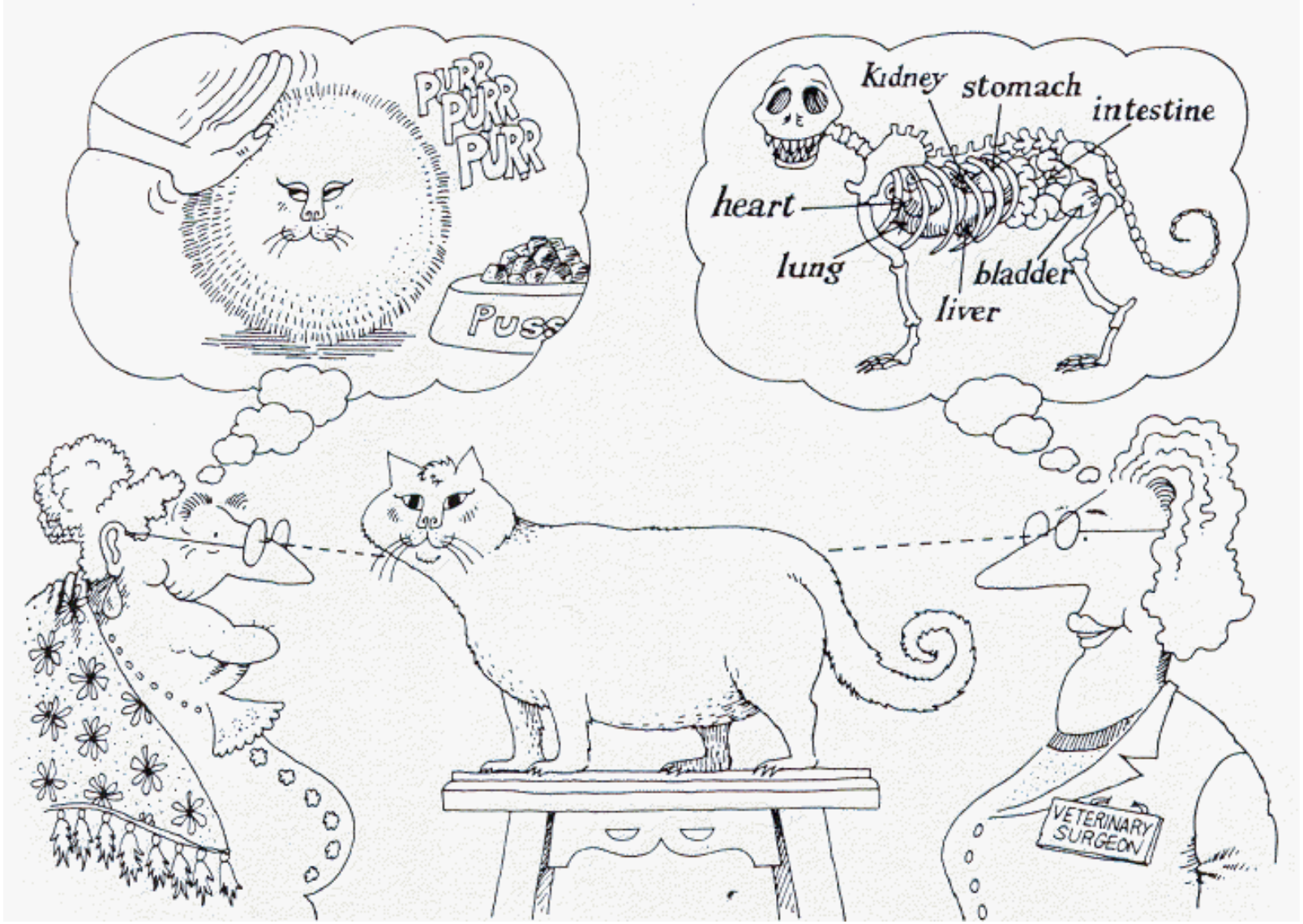
- Estes princípios não são recentes, mas,
  - ◊ no Modelo de Objectos eles são agrupados de uma forma sinergética ...
    - · ... onde o todo é superior à soma das partes

Paulo Trigo Silva

Abstracção

- "... *denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer*"
  - ◊ concentrar no essencial / ignorar o acessório
  - ◊ o foco é naquilo que o objecto faz e não como será implementado
- Existe uma gama de abstracção que vai,
  - ◊ desde o objecto que está "perto do mundo real"
  - ◊ até aqueles que "não têm qualquer contrapartida no mundo real"
- Abstracções nessa gama (primeiro as mais úteis ...)
  - ◊ entidade / um conceito do "mundo"
  - ◊ acção / objecto com diversas operações todas funcionalmente iguais
  - ◊ máquina virtual / objecto que agrupa operações usadas por um mais alto
  - ◊ acidental / objecto que empacota operações sem relação entre elas

Abstracção (cont.)



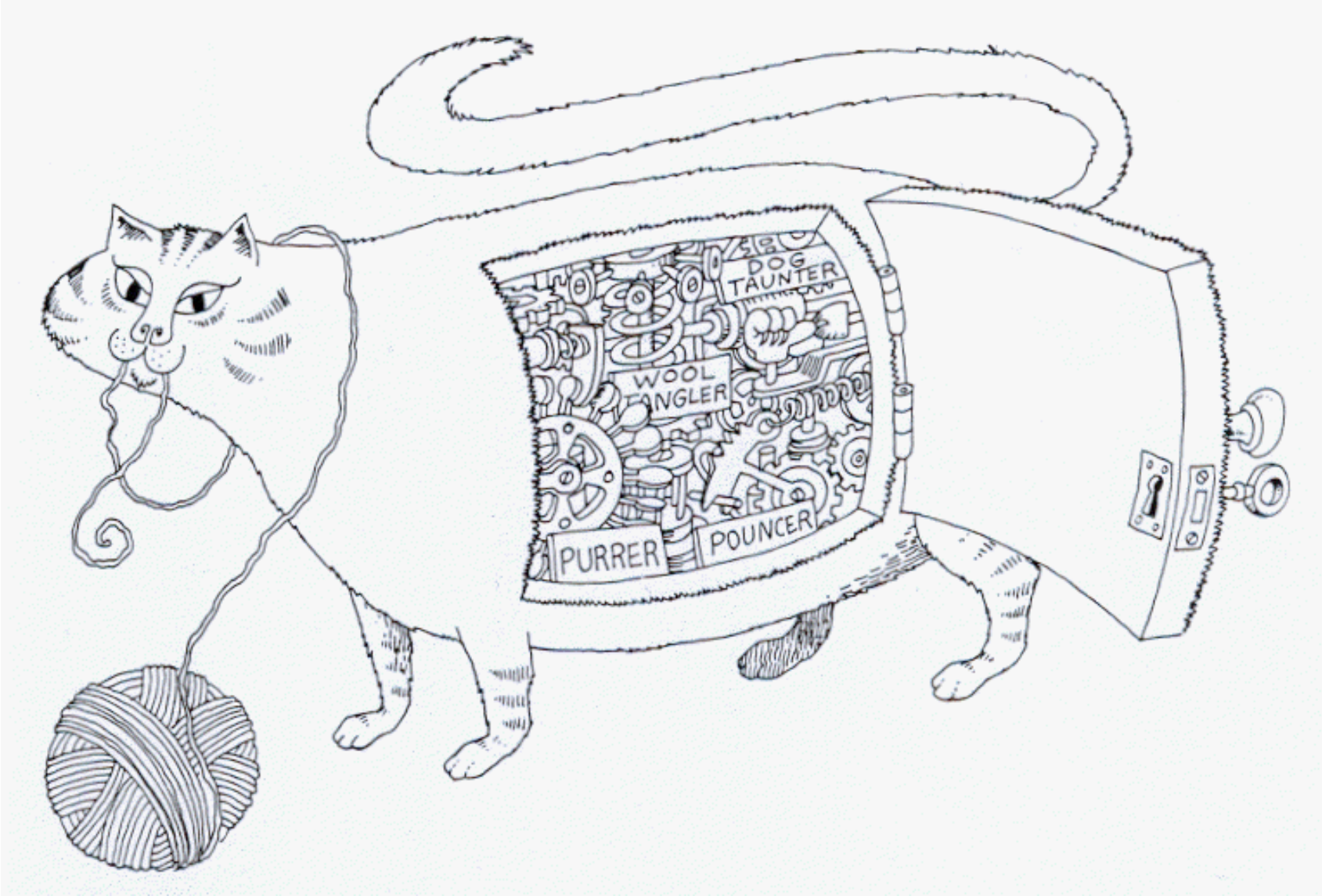
Paulo Trigo Silva

Encapsulamento

- "... *process of hiding all of the details of an object that do not contribute to its essential characteristics*"
  - ◊ também designado por *information hiding*
  - ◊ isolar o que do objecto deve ser visível do exterior (a sua interface)
  - ◊ dos aspectos internos de implementação
- Abstracção e Encapsulamento são conceitos complementares,
  - ◊ Abstracção
    - · o que vê quem olha de fora ...
  - ◊ Encapsulamento
    - · evita que quem use veja como a abstracção foi implementada ...
- Encapsulamento fornece
  - ◊ barreiras explícitas entre diferentes Abstracções ...
  - ◊ "... *for abstraction to work, implementation must be encapsulated*"

Paulo Trigo Silva

Encapsulamento (cont.)



Paulo Trigo Silva

Modularidade

- "... *property of a system that has been decomposed into a set of cohesive and loosely coupled modules*"
  - ◊ tradicionalmente tem a ver com o agrupamento de sub-programas
  - ◊ na perspectiva da orientação por objectos tem a ver com
    - · decisões de empacotamento físico de classes e objectos lógicos
- ... na construção de Módulos
  - ◊ aumentar a coesão – agrupar Abstracções logicamente relacionadas ...
  - ◊ diminuir o acoplamento – minimizar dependências entre Módulos ...
  - ◊ ... identificar Classes e Objectos – desenho lógico
  - ◊ ... organizar Classes e Objectos em Módulos – desenho físico
- Abstracção, Encapsulamento e Modularidade, são sinergéticos ...
  - ◊ "... *an object provides a crisp boundary around a single abstraction,*
  - ◊ *and, both encapsulation and modularity provide barriers around this abstraction*"

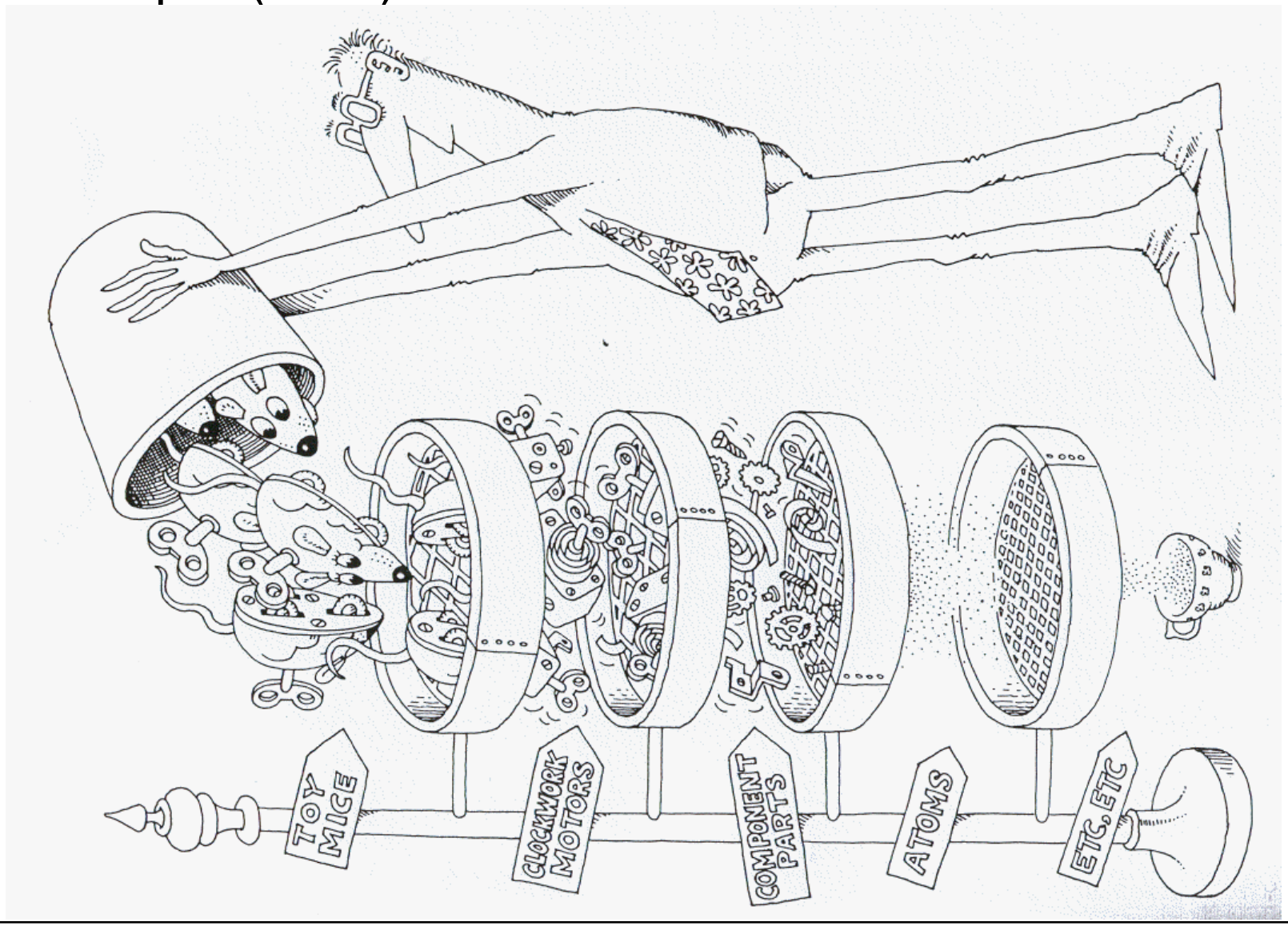
Modularidade (cont.)

Paulo Trigo Silva

Hierarquia

- "... *hierarchy is a ranking or ordering of abstractions*"
  - ◊ Abstracção é muito útil, mas podemos encontrar muitas mais abstracções do que aquelas possíveis de entender numa única vez
  - ◊ Encapsulamento e Modularidade ajudam a gerir a complexidade
  - ◊ ... mas não são suficientes !
- ... as Abstracções podem formar Hierarquias
  - ◊ identificar as Hierarquias simplifica a compreensão do problema
- As 2 Hierarquias mais importantes são
  - ◊ a de Classes (*kind-of*) – herança simples e múltipla
  - ◊ e a da sua estrutura de Objectos (*part-of*) – agregação
- ... com Herança e Agregação constróem-se "níveis de Abstracção"
  - ◊ Herança – "árvore" de refinamento de Abstracções
  - ◊ Agregação – "junção" onde diversas Abstracções formam uma "maior"

Hierarquia (cont.)



Abstracção, Encapsulamento, Hierarquia ...

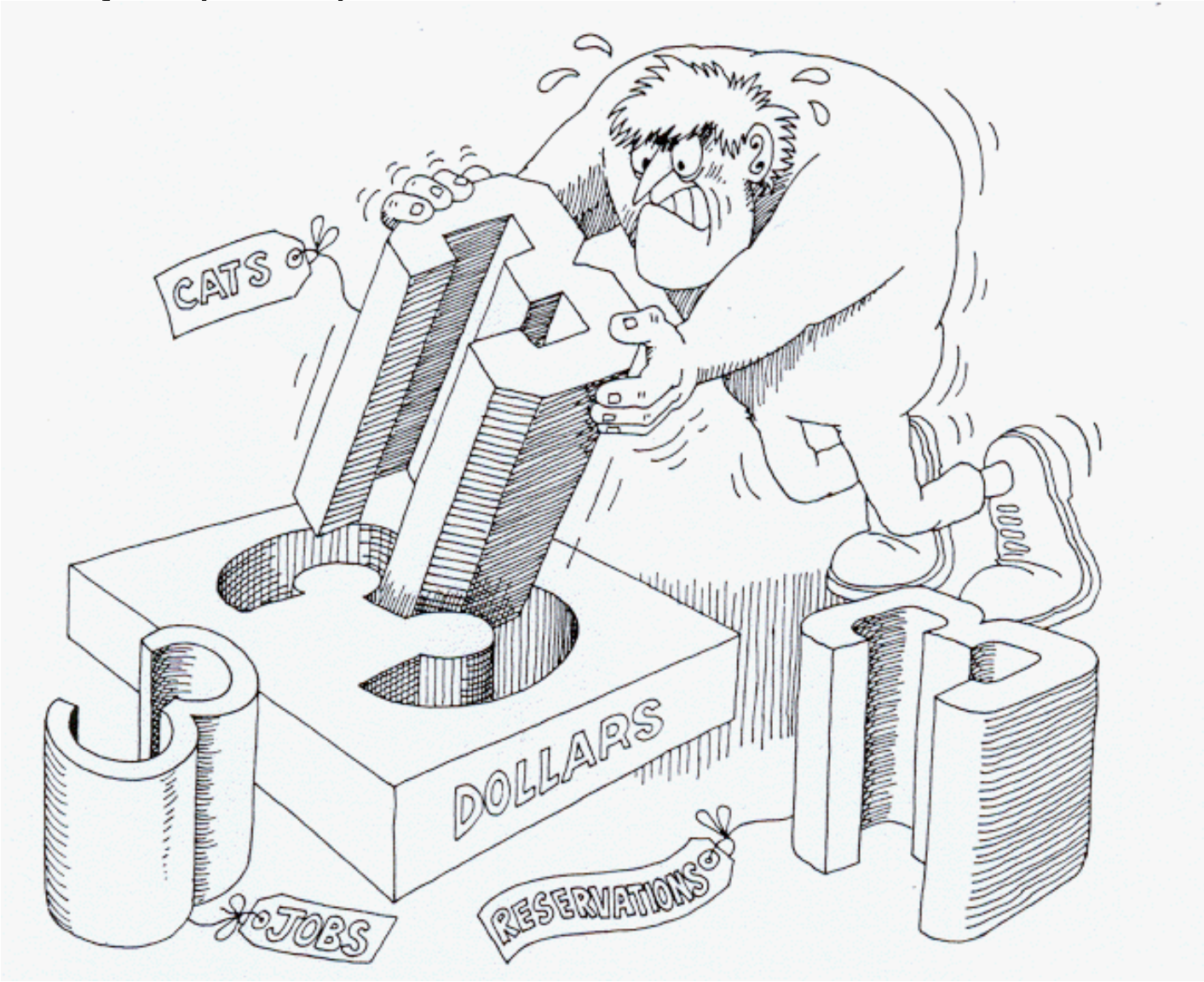
- Entre estes conceitos
  - ◊ existe alguma "tensão" !
- "... *abstraction attempts to provide an opaque barrier, behind which methods and state are hidden*"
- "... *inheritance requires opening this interface and may allow state as well as methods to be accessed without abstraction*"
- "... *with inheritance, encapsulation can be violated in three ways:*"
  - ◊ *subclass might access an instance variable of its super-class;*
  - ◊ *call a private operation of its super-class;*
  - ◊ *refer directly to super-classes of its super-class*"
- Diferentes linguagens tratam estas questões de diferente modo
  - ◊ interface da classe com blocos *private*, *protected*, *public* – C++, Java

Paulo Trigo Silva

Tipificação

- "... *is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways*"
  - ◊ uma Classe "não é um Tipo"
    - · ... uma Classe implementa um Tipo
  - ◊ um Tipo "é uma interface" (no Java isso é explícito; no C++ é implícito)
    - · ... diferentes Classes podem implementar essa interface / esse Tipo
- ... uma linguagem de programação pode ter Tipificação
  - ◊ forte (*strong*) – expressões são consistentes com o tipo a que pertencem
    - · *Java*, *C++*, *Object Pascal*, ...
  - ◊ fraca (*weak*) – as expressões não têm um tipo – *Lisp*, *Prolog*, ...
  - ◊ *strong typing* é diferente de *static typing*
    - · *static typing* (ou *static binding* ou *early binding*) – tipo das variáveis e expressões são fixados em tempo de compilação
    - · *dynamic binding* ou *late binding* – ... em tempo de execução

Tipificação (cont.)

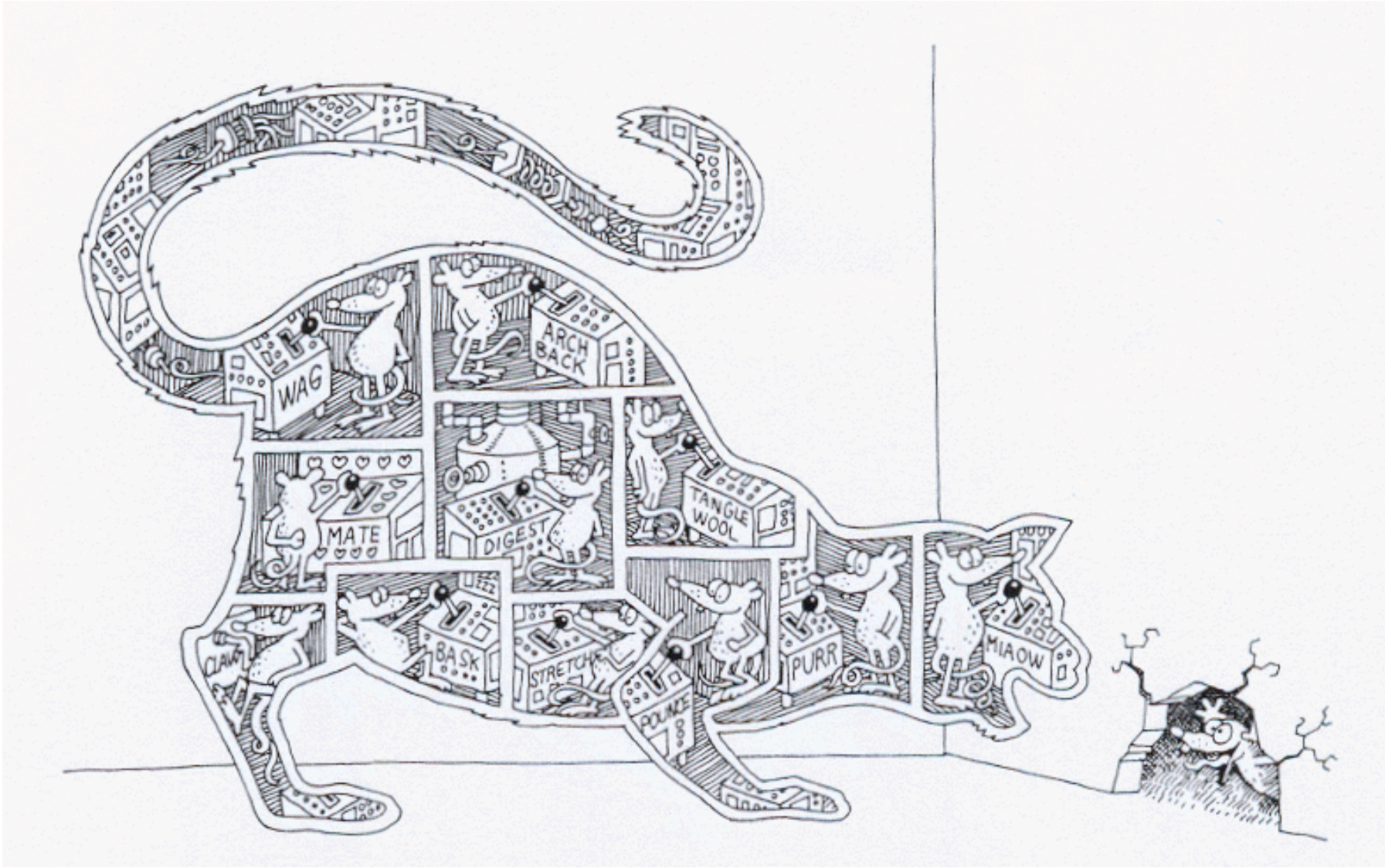


Paulo Trigo Silva

Concorrência

- "... *is the property that distinguishes an active object from one that is not active*"
  - ... tratamento simultâneo de diferentes eventos
  - cada programa tem pelo menos um fluxo de controlo (*thread of control*)
  - um sistema concorrente pode ter diversas *threads of control*
- Objecto é um conceito que unifica duas perspectiva
  - Abstracção do mundo real
  - Abstracção do comportamento de um processo separado
    - estes objectos representam fluxo(s) de controlo independente(s) ...
    - são designados por objectos activos (*active objects*)
- O "mundo pode ser visto" como
  - um conjunto de objectos que cooperam,
  - alguns dos quais são activos, e como tal,
  - actuam como centros de actividade independente ...

Concorrência (cont.)

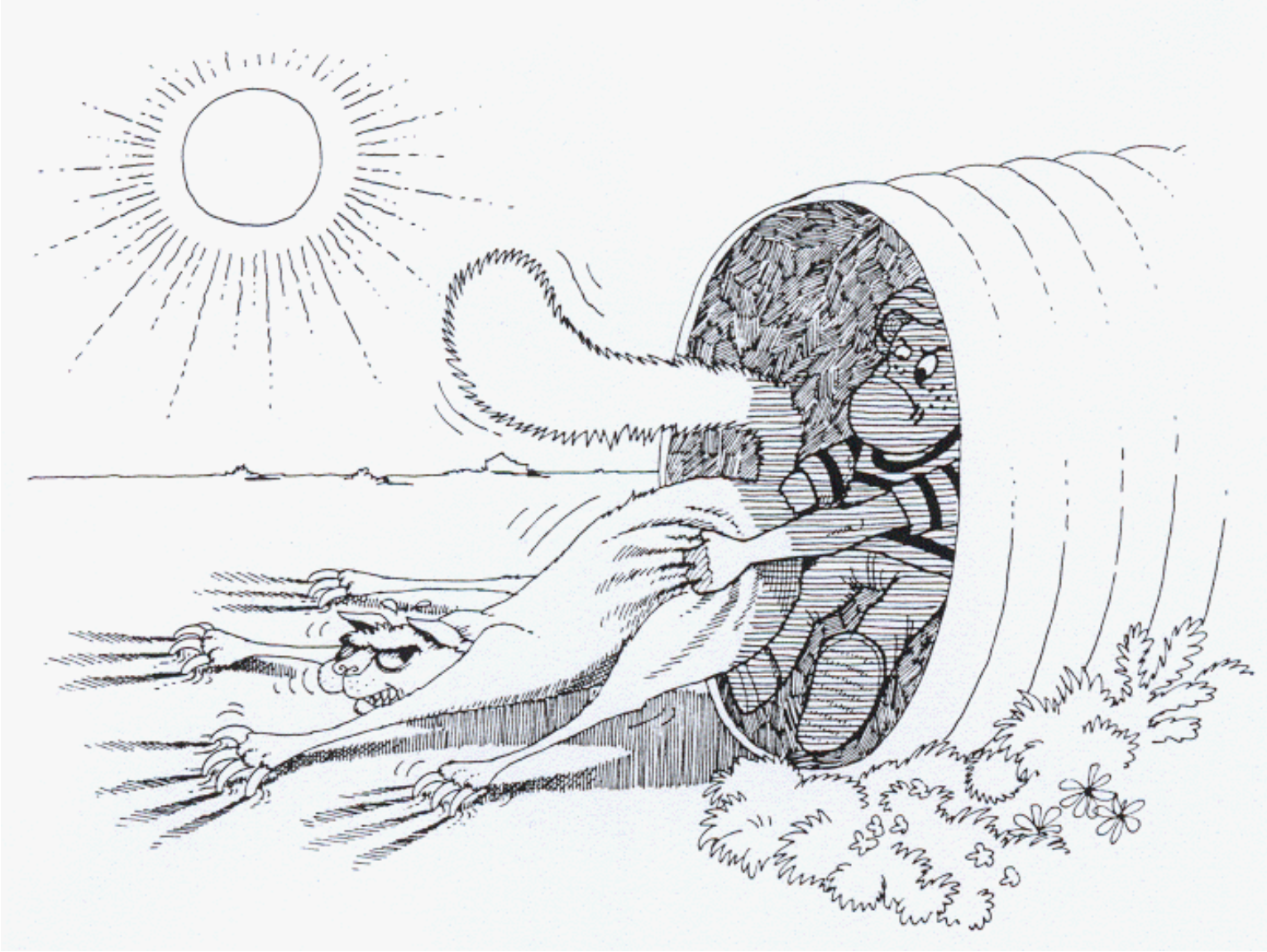


Paulo Trigo Silva

Persistência

- "... *is the property of an object through which its existence transcends time (i.e. the object continues to exist after its creator ceases to exist) and/or space (i.e. the object location moves from the address space in which it was created)*"
  - ◊ ... manter as características e poder migrar entre ambientes diferentes
- Há um leque de níveis de persistência
  - ◊ resultado da avaliação de uma expressão; variáveis locais em procedimentos; variáveis globais;
    - · ... tratados pelas tecnologias das linguagens de programação
  - ◊ dados entre execuções de um programa; dados entre versões de um programa; dados independentes dos programas
    - · ... tratados pelas tecnologias das bases de dados
- A introdução da Persistência no Modelo de Objectos
  - ◊ está na origem das Bases de Dados Orientadas a Objectos ...

Persistência (cont.)



Paulo Trigo Silva

... o que é e o que não é um Objecto ?

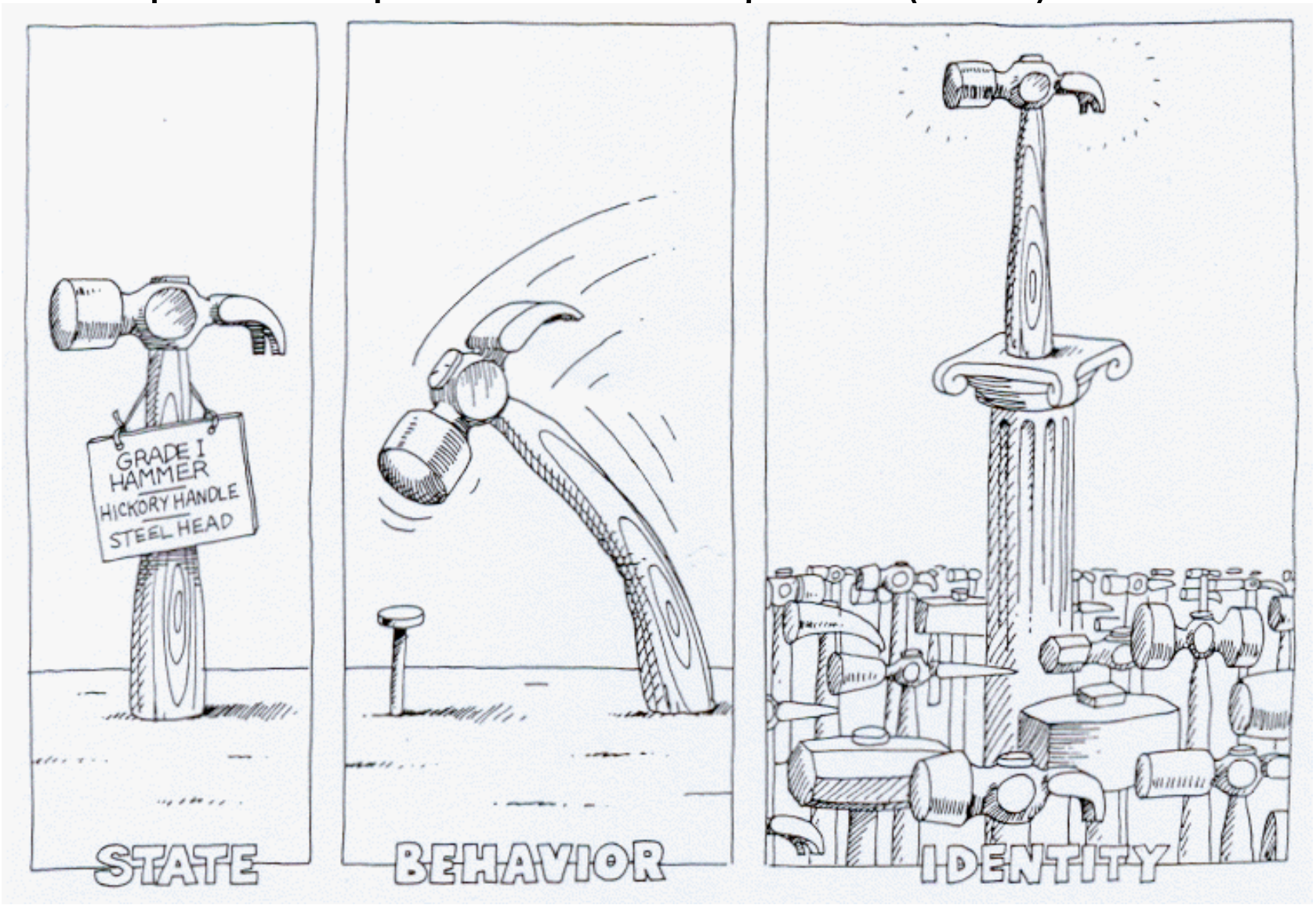
- "... *an object has a state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable"*

- Na perspectiva da cognição humana, um Objecto é,
  - ◊ uma coisa tangível (que se pode pegar) ou visível
  - ◊ um conceito que seja possível compreender
  - ◊ algo que seja alvo de acções ou pensamentos

- Isto expande um pouco a noção de Objecto
  - ◊ não apenas algo com existência no "mundo real", mas também,
  - ◊ "colaborações" entre objectos que formem comportamento diferenciado

- ... tudo no mundo são Objectos ... mas algumas são menos !
  - ◊ o tempo, a beleza, a cor, o sentimento, a emoção,
  - ◊ são potenciais propriedades/atributos de outros objectos ...

... o que é e o que não é um Objecto ? (cont.)



Paulo Trigo Silva

Estado (*State*)

- "... *the state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties"*

- Um exemplo – "uma máquina de venda automática de bebidas aceita moedas e depois permite seleccionar uma bebida, no entanto se pretender seleccionar uma bebida sem ter colocado as moedas nada acontece ..."

- O comportamento de um Objecto é influenciado pelo "seu passado"
  - ◊ este comportamento "dependente do tempo" justifica-se,
  - ◊ pela existência de estado no Objecto,
    - dinâmico – "quantidade de moedas já aceite"
    - estático – "numero de série da máquina"

- ... ter Estado implica que cada Objecto tem que ocupar espaço ...

Comportamento (*Behavior*)

- "... *behavior is how an object acts and reacts, in terms of its state changes and message passing … for our purposes the terms operation and message are interchangeable"*
  - ◊ ... o comportamento é completamente definido pelas acções do objecto
- ... objecto desencadeia uma acção com o objectivo de obter reacção
  - ◊ por exemplo, um objecto invoca operações de *push* e *pop*
    - · para adicionar ou eliminar um elemento de um objecto *stack* ...
- Geralmente uma mensagem é
  - ◊ apenas uma operação que um objecto executa sobre outro
    - · em *Smalltalk* diz-se que o "objecto envia mensagens"
  - ◊ a mensagem pode "passar espaços de endereçamento", *RPC*, *RMI* ...
- As operações disponibilizadas por um objecto
  - ◊ são geralmente declaradas como métodos (*method*) – *C++*, *Java*, *CLOS*

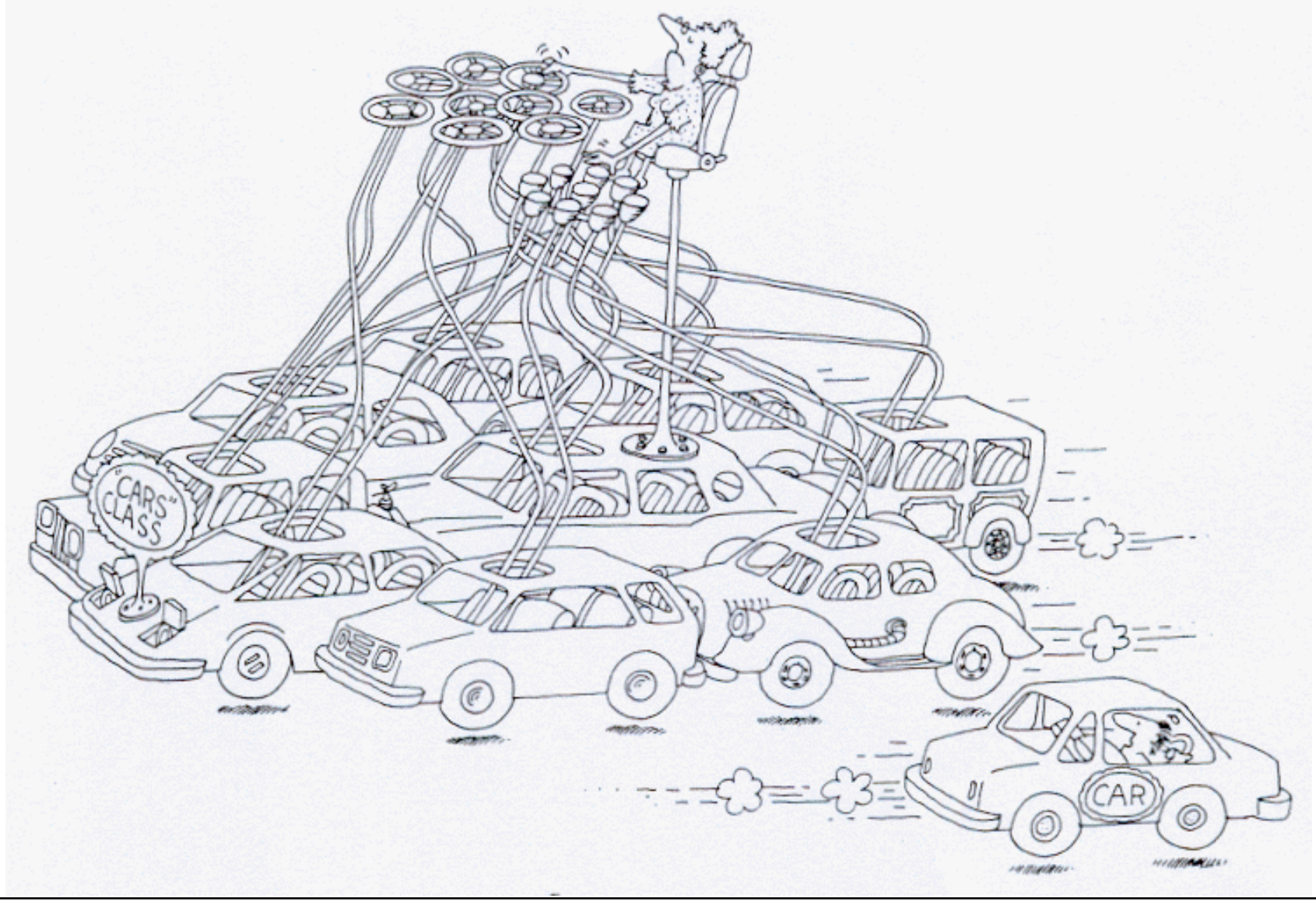
Identidade (*Identity*)

- "... *identity is that property of an object which distinguishes it from all other objects*"
  - ◊ como cada objecto é distinguível de todos os outros, a sua identidade é preservada mesmo quando o seu estado é completamente alterado
  - ◊ ... uma antiga questão Zen – "*is a river still the same river from one day to the next, even though the same water never flows through it ?*"
- ... mescla entre Identidade e Endereçamento (*Addressability*)
  - ◊ em muitas linguagens de programação os objectos se distinguem
    - · através do nome das variáveis
- ... mescla entre Identidade e Valor de Dados (*Data Value*)
  - ◊ em muitos sistemas de bases da dados os objectos se distinguem
    - · pelos valores de alguns dos seus atributos (chaves)

... o que é e o que não é uma Classe ?

- "... *a class is a set of objects that share a common structure and a common behavior*"
  - ◊ um objecto é uma entidade concreta que existe no tempo e no espaço
  - ◊ uma Classe representa apenas uma Abstracção, a essência do objecto
- Não é uma Classe ...
  - ◊ um Objecto não é uma Classe
    - · embora uma Classe possa ser um Objecto ...
  - ◊ um conjunto de Objectos que não partilhe estrutura ou comportamento não pode ser abstraído numa Classe
    - · apenas partilham o facto de serem Objectos ...
- Pode-se ver uma Classe como um contrato
  - ◊ entre uma Abstracção e todos os seus clientes
  - ◊ esta perspectiva de contrato distingue entre visão externa e interna
    - · interface (visão externa) e implementação (visão interna) da Classe

... o que é e o que não é uma Classe ? (cont.)

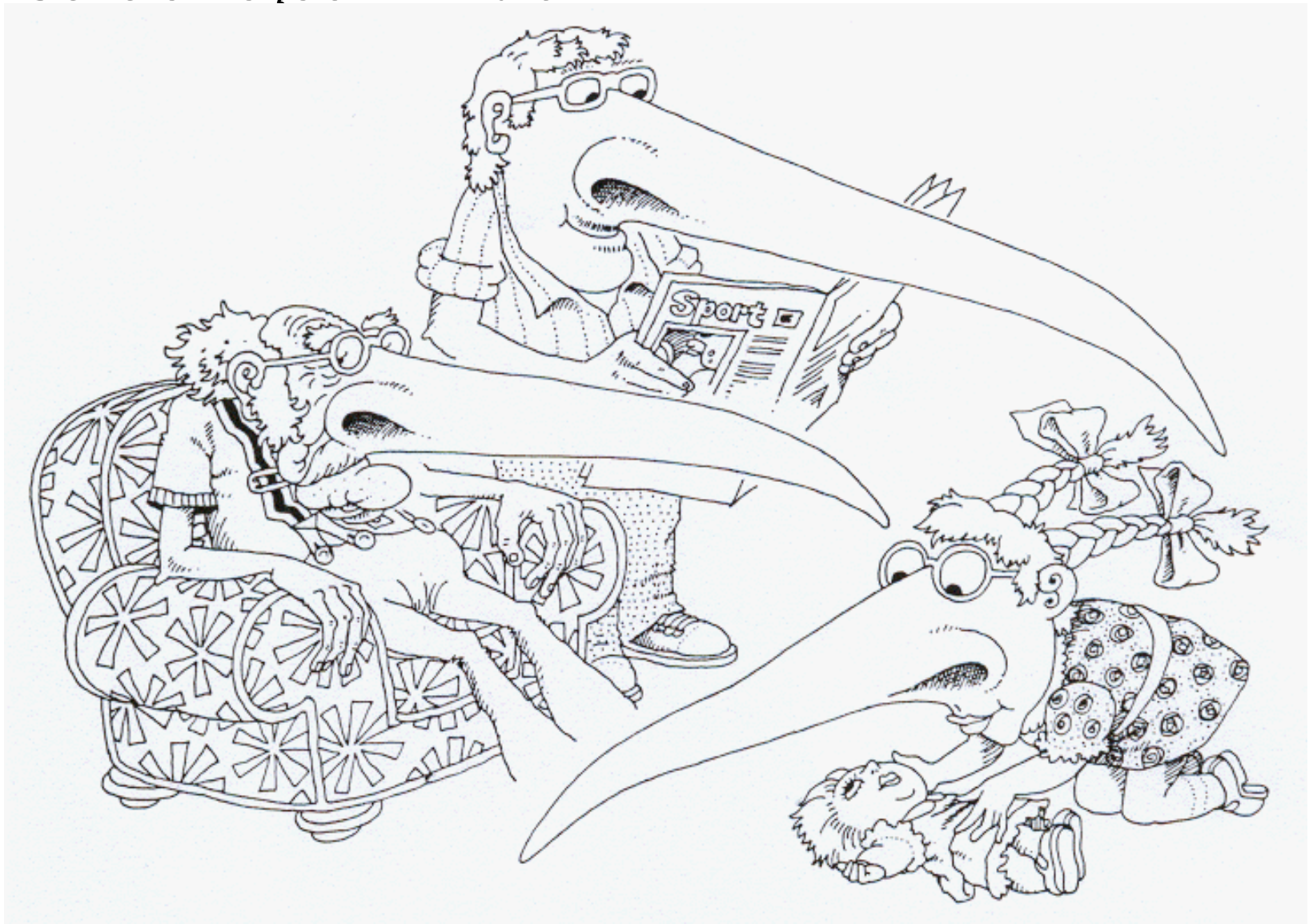


Paulo Trigo Silva

Relações entre Classes

- Quais as semelhanças e diferenças entre os seguintes elementos ?
  - ◊ flores, margaridas, rosas vermelhas, rosas amarelas, pétalas
- ... podemos fazer observações como,
  - ◊ margarida é um tipo de flor; rosa é um outro tipo de flor
  - ◊ rosa vermelha e rosa amarela são ambas tipos de rosas
  - ◊ pétala é uma parte de ambos os tipos de flor
- Existem 3 formas de Relações
  - ◊ Generalização – "*kind of*"
    - · rosa é uma subclasse especializada da classe mais geral flor
  - ◊ Agregação – "*part of*"
    - · uma pétala é uma parte de uma flor
  - ◊ Associação – "*semantic connection among otherwise unrelated classes*"
    - · rosas e floristas – independentes mas podem colaborar num negócio

Generalização – "*kind of*"



... outra forma de Relação

- Existe ainda 1 outra forma de Relação, designada por Delegação
  - ◊ ... objecto intercepta mensagens e reencaminha-os para outros objectos
  - ◊ exemplo de concretização – registo de *event handlers* em Java
  - ◊ suporte directo – *actor programming languages*, como Actors, SALSA, ...
- ... a perspectiva de Grady Booch, em "Object Oriented Design" é
  - ◊ "... *an alternative approach to inheritance involves a language mechanism called delegation, in which objects are viewed as prototypes (also called exemplars) that delegate their behavior to related objects, thus eliminating the need for classes"*
- Generalização concretizada por Agregação e Delegação
  - ◊ uma super-classe com várias sub-classes pode ser concretizada como
    - · agregação na super-classe – classe "junção", das várias sub-classes
  - ◊ herança de operações não é automática, pelo que,
    - · o objecto "junção" recebe mensagens e delega-as ao objecto destino

"Medir" a qualidade de uma Abstracção

- Excepto em situações muito simples
  - ◊ "nunca", à primeira, sai a definição correcta e completa de uma Classe !
- A definição de Classes e Objectos
  - ◊ é um processo iterativo e incremental
  - ◊ ... e esse refinamento isso tem um custo em termos de
    - · recompilações, compreensão e integridade do sistema ...
- Como se sabe se uma Classe ou Objecto está bem desenhado ?
  - ◊ ... 5 métricas úteis:
    - · Acoplamento (*Coupling*)
    - · Coesão (*Cohesion*)
    - · Suficiência (*Sufficiency*)
    - · Plenitude (*Completeness*)
    - · Primitividade (*Primitiveness*)

Conceitos "importados" do Desenho Estruturado ... com "interpretação liberal" também se aplicam ao Modelo de Objectos

Paulo Trigo Silva

Acoplamento (*Coupling*)

- Definição de Acoplamento por Stevens, Myers e Constantine,
  - "... *the measure of the strength of association established by a connection from one module to another.*"
  - "*Strong coupling complicates a system since a module is harder to understand, change or correct "if it is highly interrelated with other modules*"
  - "*Complexity can be reduced by designing systems with the weakest possible coupling between modules*"
- Acoplamento em O.O refere-se sobretudo a Classes e Objectos
  - embora o que respeita aos Módulos também seja aplicável em O.O.
- ... alguma "tensão" entre conceitos de Acoplamento e Generalização
  - um fraco Acoplamento entre Classes é desejável, mas,
  - a Generalização é um forte Acoplamento entre Classes que permite
    - explorar a partilha (factorização) entre diversas Abstracções ...

Coesão (*Cohesion*)

- Definição de Coesão por Grady Booch,
  - ◊ "... *measures the degree of connectivity among the elements of a single module*"
  - ◊ "… *and for object-oriented a single class or object*"
- A forma menos desejável de coesão é a "acidental"
  - ◊ *coincidental cohesion* – onde Abstracções sem qualquer relação se "atiram" para a mesma Classe ou Módulo ...
    - · uma Classe que represente os conceitos de Gato e Automóvel !
- A forma mais desejável de coesão é a "funcional"
  - ◊ *functional cohesion* – onde todos os elementos de uma Classe ou Módulo "trabalham" em conjunto para o mesmo comportamento ...
    - · a Classe Aluno é funcionalmente coesa se a sua semântica abrange o comportamento total do aluno, e nada mais do que o aluno ...

Suficiência (*Sufficiency*)

- Definição de Suficiência por Grady Booch,
  - "... *the class or module captures enough characteristics of the abstraction to permit meaningful and efficient interaction.*"
  - "*To do otherwise renders the component useless*"
- Por exemplo, ao desenhar a classe "Conjunto"
  - é essencial considerar uma operação de remoção de um elemento
  - mas não se poderá usar a classe sem a operação de juntar elemento !
- Na prática,
  - as transgressões a esta característica são detectadas muito cedo
  - elas aparecem sempre que um cliente precisa utilizar a Abstracção

Paulo Trigo Silva

Plenitude (*Completeness*)

- Definição de Plenitude por Grady Booch,
  - ◊ "... *the interface of the class or module captures all the meaningful characteristics of the abstraction*"
- Suficiência e Plenitude
  - ◊ suficiência implica que a interface seja a mínima possível
  - ◊ plenitude implica cobrir todos os aspectos de uma abstracção
- Plenitude é uma característica a abordar com cautela !
  - ◊ e a não exagerar ...
- Oferecer todas as operações possíveis para uma abstracção,
  - ◊ sobrecarrega o utilizador de informação
  - ◊ pode ser desnecessário pois operações de alto nível se podem decompor em outras mais elementares
    - · por isso se sugere a Primitividade das Classes ...

Primitividade (*Primitiveness*)

- Definição de Primitividade por Grady Booch,
  - "... *primitive operations are those that can be efficiently implemented only if given access to the underlying representation of the abstraction*"
- Por exemplo, na classe "Conjunto" – adicionar elemento é primitivo
  - porque implementar esta operação, implica,
  - ter acesso à representação interna do conjunto
- ... mas, adicionar quatro elementos de uma vez não é primitivo
  - pode-se implementar sobre a operação primitiva de adicionar só um ...
- ... mas, é também necessário considerar questões de desempenho
  - uma operação que se possa implementar sobre outras primitivas
  - mas à custa de pesado esforço computacional
  - que poderia ser reduzido se ela fosse uma operação primitiva
  - ... é também candidata a tornar-se uma operação primitiva

Heurísticas para escolher Operações

- "... *a good designer knows how to find the appropriate balance between too much contracting, which produces fragmentation, and too little, which yields unmanageably large modules*"

- Em que Classe colocar as operações ? – alguns critérios ...
  - ◊ Reutilização
    - · este comportamento é útil em mais do que um contexto ?
  - ◊ Complexidade
    - · qual a dificuldade em implementar o comportamento ?
  - ◊ Aplicabilidade
    - · que relevância tem o comportamento no tipo em que for colocado ?
  - ◊ Conhecimento da Implementação
    - · a implementação do comportamento depende dos detalhes internos de determinado tipo ?

Paulo Trigo Silva

Heurísticas para escolher Relações

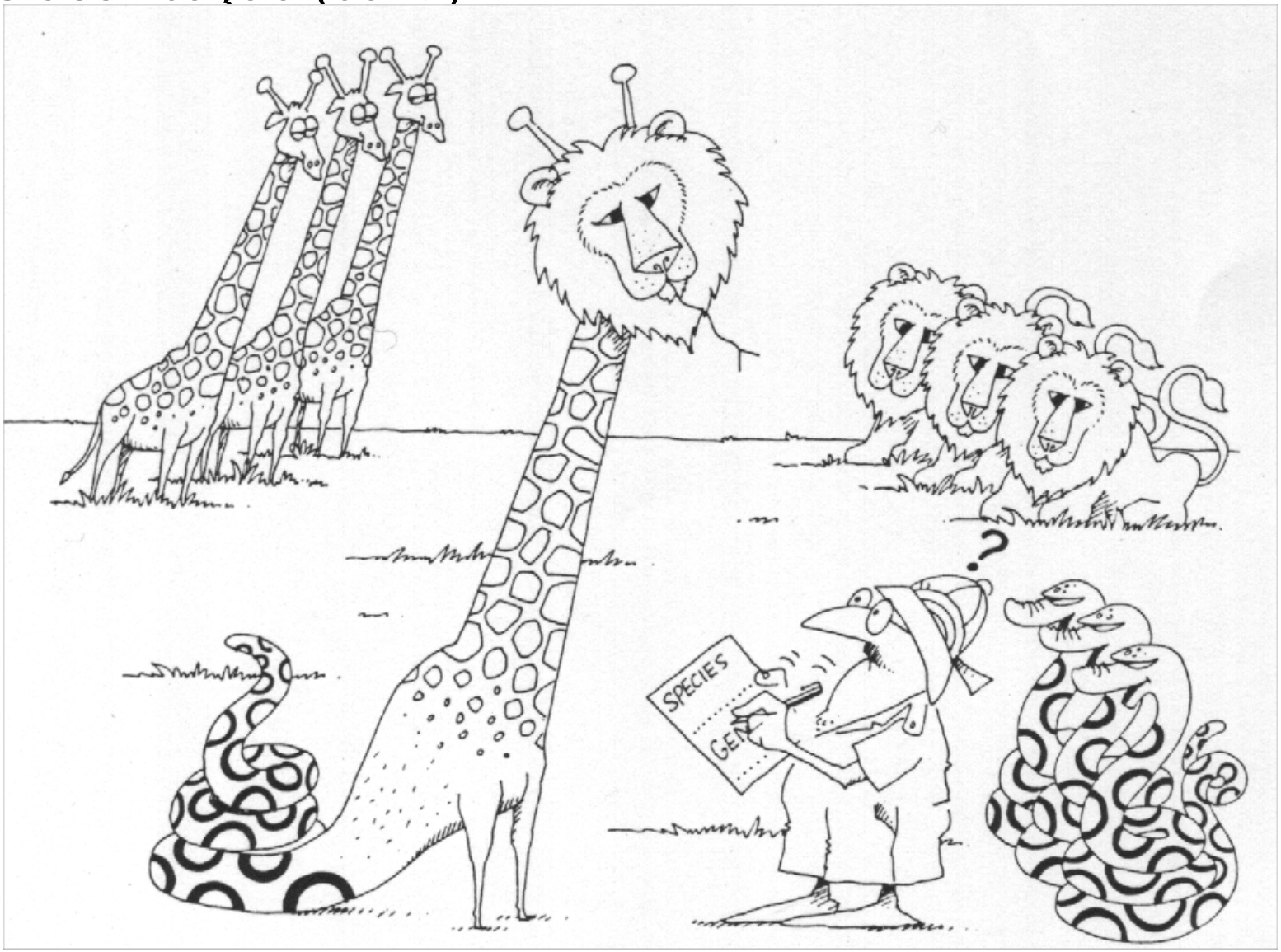
- Uma indicação – conhecida por *Law of Demeter*
  - "... *the methods of a class should not depend in any way on the structure of any class, except the immediate (top level) structure of his own class.*"
  - "*Further, each method should send messages to objects belonging to a very limited set of classes only*"
- A ideia essencial nesta indicação é a de criar Classes
  - com fraco Acoplamento
  - cujos segredos de implementação estão Encapsulados
- Generalização ou Associação ?
  - "… *inheritance is appropriate if every instance of class B may also be viewed as an instance of class A*", Meyer
  - "… *if the behavior of an object is more than the sum of its individual parts, then creating a using relationship rather than an inheritance relationship between appropriate classes is probably superior*", Booch

Classificação (*Classification*)

- O mais difícil na O.O. é identificar Classes e Objectos, que envolve,
  - descoberta
    - reconhecer abstracções
  - invenção
    - generalizar abstracções
    - construir os mecanismos de colaboração entre objectos
  - ... descoberta e invenção são ambas questões de Classificação ...
- ... e Classificação é uma questão de
  - encontrar semelhanças de estrutura e comportamento
- Por que a Classificação ?
  - "*Classification helps us to identify generalizations, specialization, and aggregation hierarchies among classes*"
  - "*Recognizing common patterns of interaction among objects, we come to invent the mechanisms that serve as the soul of our implementation*"

Paulo Trigo Silva

Classificação (cont.)

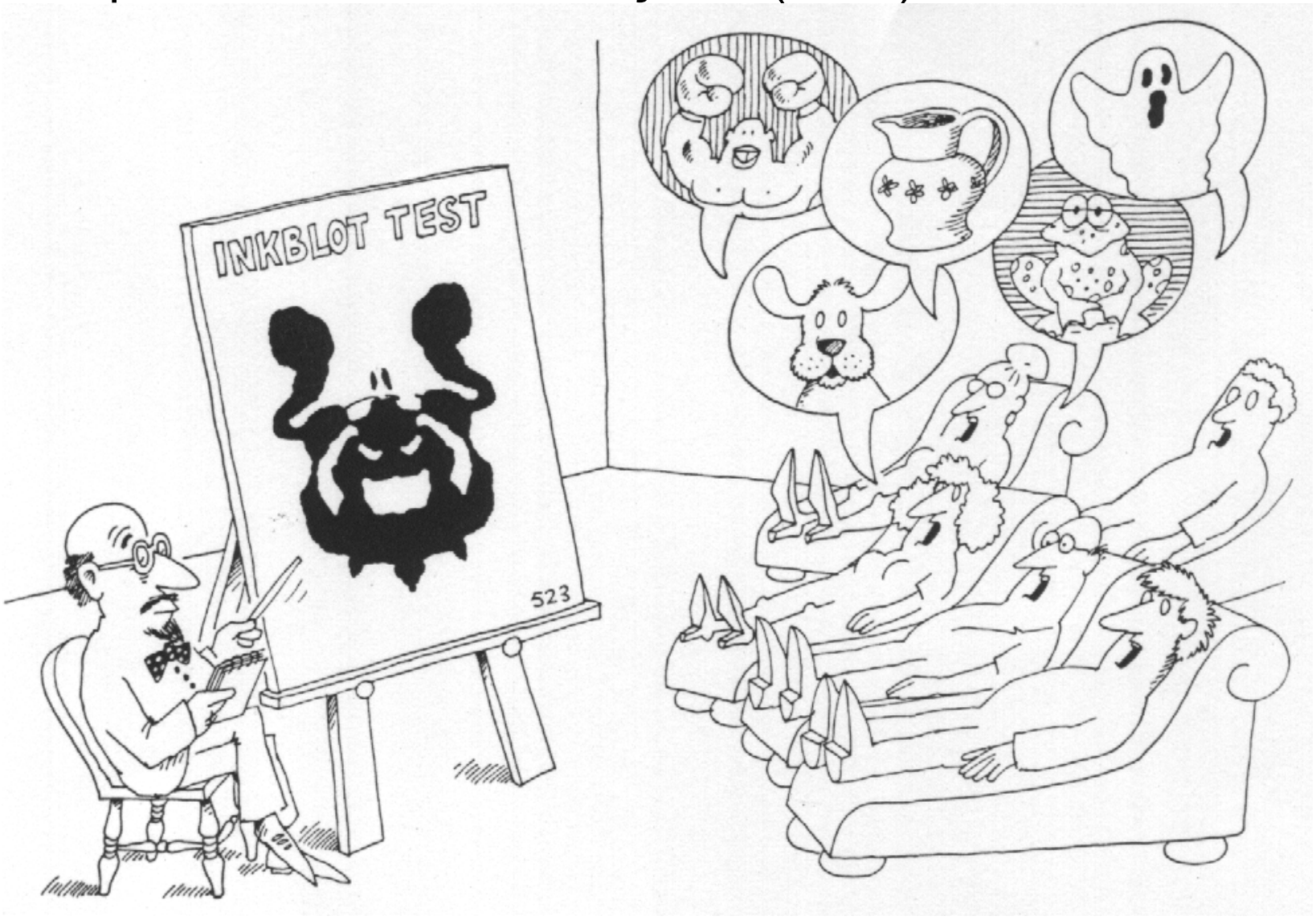


Paulo Trigo Silva

Porque é difícil a Classificação ?

- Dois importantes motivos
  - ◊ não existe uma "Classificação perfeita"
    - · embora certamente umas sejam bastante melhores que outras
  - ◊ uma "boa Classificação" requer muita experiência
    - · e um grande conhecimento do problema
- Diferentes observadores, classificam o mesmo elemento
  - ◊ de diversos modos
- ... a Classificação por vezes é evidente,
  - ◊ outras "uma questão de gosto"
- ... e na maior parte das vezes, a fase crucial da análise é a do
  - ◊ reconhecimento dos padrões adequados ... senão vejamos,
    - · "qual a semelhança entre um feixe laser e um peixinho vermelho ?"
    - · "nenhum deles sabe assobiar ! ..."

Porque é difícil a Classificação ? (cont.)



Paulo Trigo Silva

Reificar Abstracções

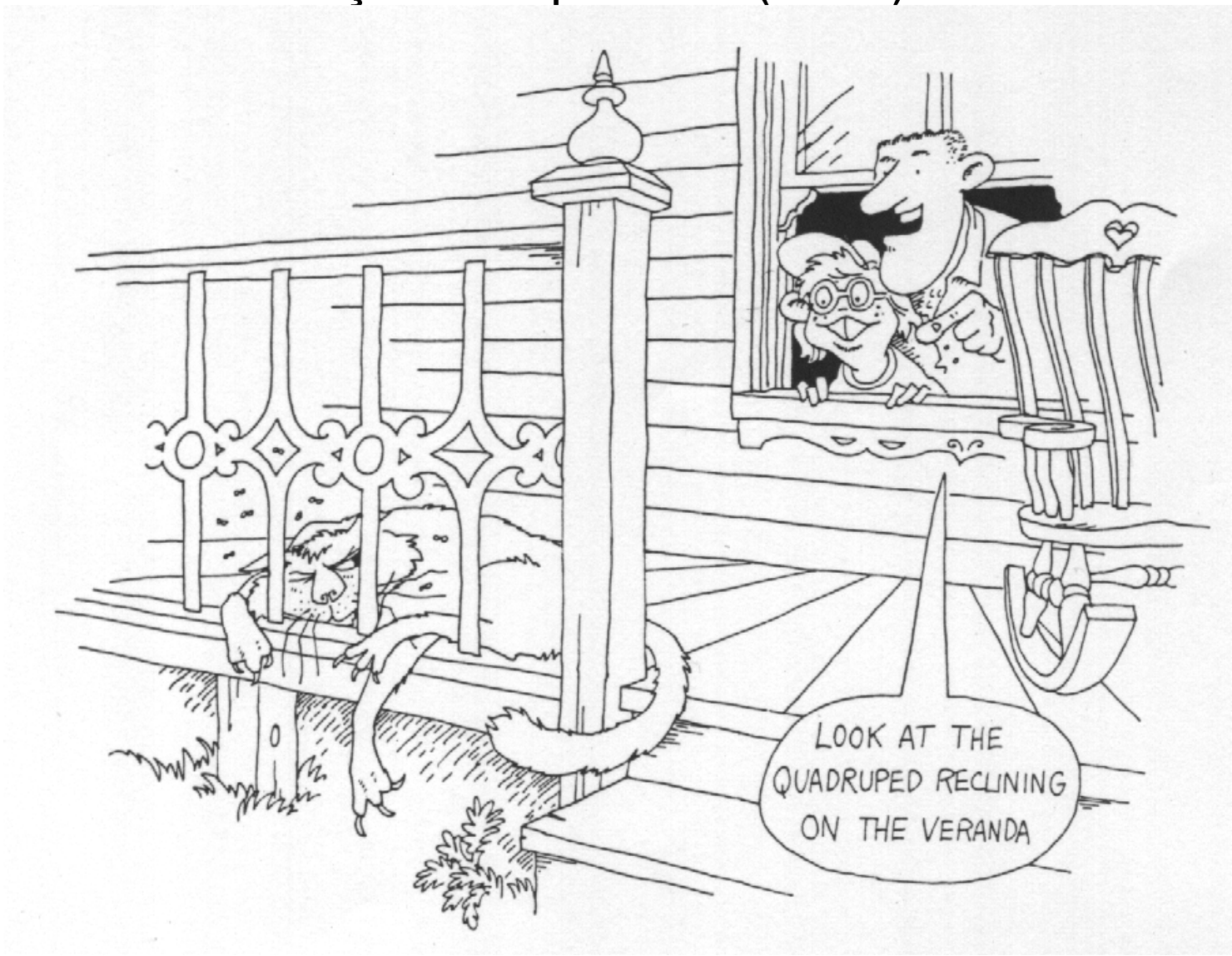
Reificar – transformar, por operação mental, conceitos abstractos em realidades concretas, em objectos; substantivar; coisificar

- Após identificar as Abstracções,
  - ◊ é necessário efectuar a sua avaliação, usando como métricas,
    - · Acoplamento, Coesão, Suficiência, Plenitude, Primitividade
- Stroustrup (C++) sugere que,
  - ◊ "*often this means that the programmer must focus on the questions:*
  - ◊ *how are objects of this class created ?*
  - ◊ *can objects of this class be copied and/or destroyed ?*
  - ◊ *what operations can be done on such objects ?*
  - ◊ *If there are no good answers to this questions, the concept probably wasn't "clean" in the first place, and it might be a good idea to think a bit more about the problem and the proposed solution instead of immediately starting to "code around" the problems*"
- Novas Abstracções devem ser colocadas em contextos já existentes
  - ◊ e isso não é *top-down* nem *bottom-up*, é incremental e iterativo ...

Nível de Abstracção adequado ...

- Classes e Objectos devem estar no nível de Abstracção adequado
  - ◊ nem muito alto, nem muito baixo ... e isso não é fácil !
  - ◊ *class promotion*
    - · mover uma classe para cima, na sua hierarquia de heranças
    - · aumento do grau de partilha
  - ◊ *grainsize conflict*
    - · uma classe muito geral pode tornar difícil herança de sub-classes
    - · grande barreira semântica (*semantic gap*)
- Regras de nomenclatura ajudam muito neste processo
  - ◊ objectos – nomes próprios: `sensorAzul`, `primeiroDaPilha`, ...
  - ◊ classes – substantivos comuns: `Sensor`, `Pilha`, `Pessoa`, ...
  - ◊ operações modificadoras – verbos: `mover`, `colocar`, `remover`, ...
  - ◊ operações selectoras – verbos da forma "ser/estar": `estaVazio`, ...
  - ◊ ... convenções também simplificam – `vazio?` em vez de `estaVazio` ...

Nível de Abstracção adequado ... (cont.)



Paulo Trigo Silva

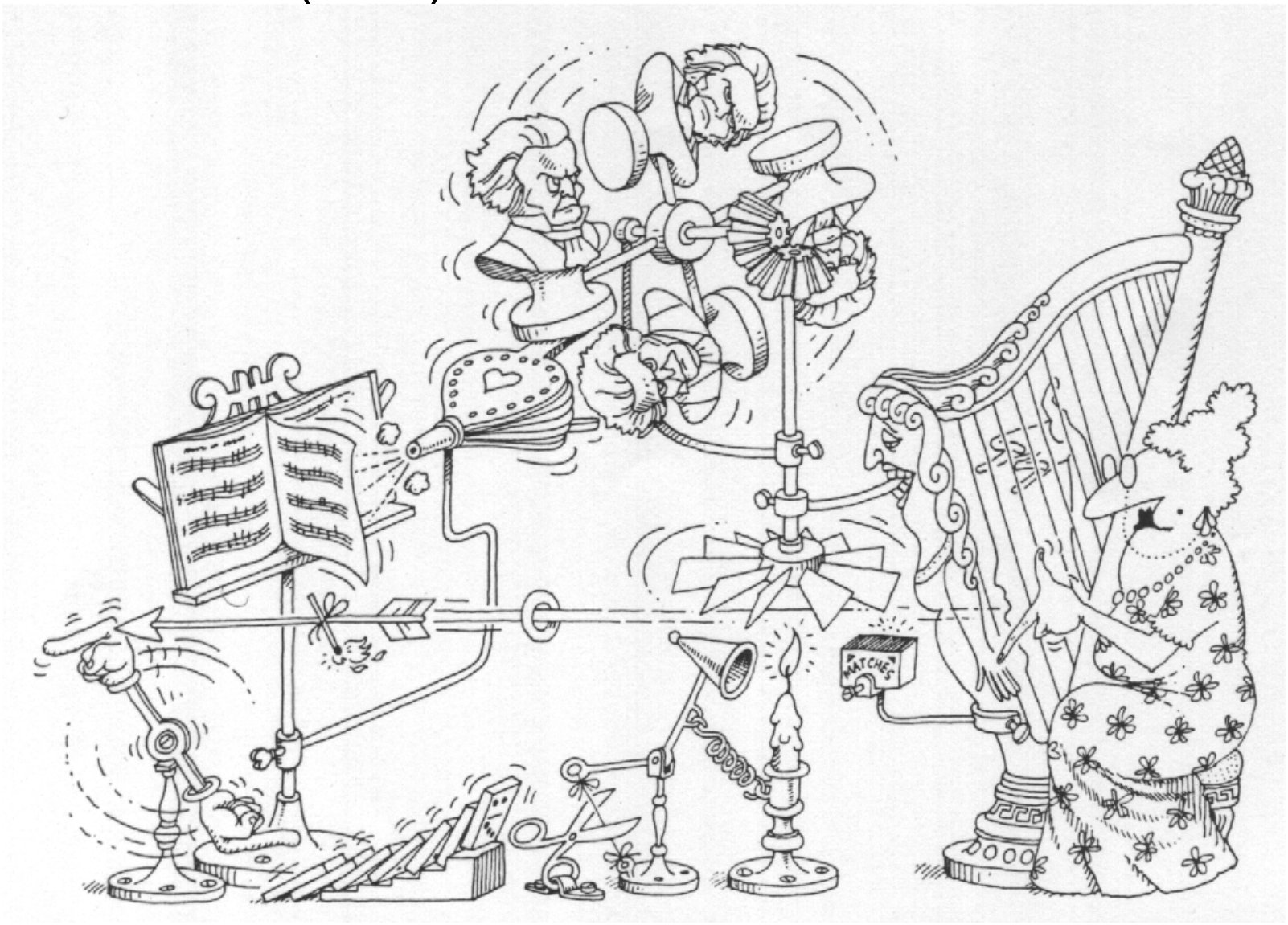
Mecanismos (*Mechanisms*)

- "… *any structure whereby objects work together to provide some behavior that satisfies a requirement of the problem*"

- Exemplo de mecanismos – usado nas interfaces gráficas
  - ◊ diversos objectos colaboram para apresentar uma imagem ao utilizador
    - · a janela; a vista; o modelo a visualizar e um cliente que sabe quando (mas não como) apresentar o modelo;
  - ◊ o cliente começa por dizer à janela que se desenhe; isso pode envolver diversas sub-vistas; a janela diz a cada uma delas para se desenhar;
  - ◊ cada sub-vista diz ao seu modelo que se desenhe, ...

- ... neste mecanismo
  - ◊ o modelo não-está-acoplado à janela e vista em que é apresentado
  - ◊ vista pode enviar mensagens ao modelo, mas não o inverso
  - ◊ Smalltalk e Java, usam uma variação deste mecanismo,
    - · MVC (*model-view-controller*)

Mecanismos (cont.)



Exemplos de Mecanismos

- Sistema Operativo
  - ◊ estrutura pode ser descrita, ao mais alto nível de abstracção, de acordo com o mecanismo adoptado para despachar (*dispatch*) processos ...
  - ◊ um desenho pode ser monolítico – DOS, incluir um núcleo – Unix, ...
- Sistema de Informação
  - ◊ estrutura pode ser descrita de acordo com o mecanismo usado para separar as várias camadas – utilização, negócio, dados, ...
- Inteligência Artificial
  - ◊ para o desenho de sistemas de raciocínio existem diversos mecanismos
  - ◊ um bastante usado é o do *blackboard*, no qual diversas fontes de conhecimento actualizam de modo independente este *backboard*;
  - ◊ neste mecanismo não existe um controlo central, mas qualquer actualização do *blackboard* pode disparar um agente que se proponha resolver determinado problema ...