“Components” – Hardware analogy

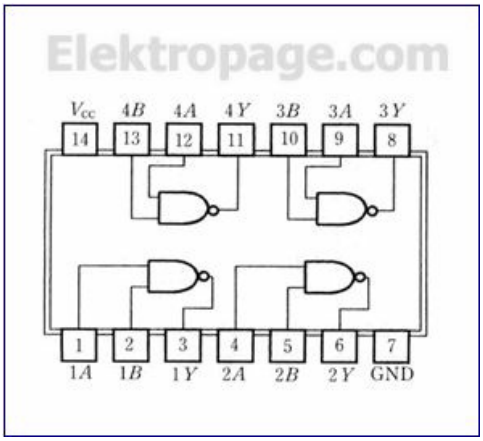
Consider a PC.
Consists of mechanical and electronic *components* (drives, boards, etc.)
held together by standard *interfaces* (e.g. PCI bus).
Boards consist of ICs conforming to standard interfaces.
ICs often consist of numerous standard cells.
Each standard cell consists of transistors.
We find a hierarchy of large component composed of smaller components.

Components are units of

- Sale
- Reusability
- Maintenance
- Documentation
- Responsibility
- Change
- Sharing
- *Design*

An “Interface”

...is a set of assumptions that the “clients” of a component may make about how to “use” the component.
Consider a particular IC – a 7400 TTL in a DIP package, which is a ‘quad NAND-gate’.



Aspects of its interface

- *Physically it has 14 pins in precise physical positions.*
- *Pin 7 is ground. Pin 14 is 5VCC.*
- *Voltage levels are as defined in the TTL standard.*
- *Pins 3, 6, 8, 11 are outputs.*
- *All other pins are inputs.*

In a sense the above are “syntactic” aspects of the interface.

Syntactic Interface and Semantic Interface

So far all aspects of the interface are shared by other ICs.
One more requirement:
The output on pin 3 is (after a given maximum delay) the NAND of the inputs on pins 1 and 2. And likewise for pins 6, 4, & 5, pins 8, 9, & 10, and pins 11, 12, & 13.
This defines the "semantics" of the interface and distinguishes the 7400 from e.g. the 7408 – "AND".
Syntax — form.
Semantics — meaning.
The use of a part with the wrong *syntactic* interface can be detected with a minimum of intelligence.

- E.g. The part won't fit. Two outputs connected together. Voltage levels are incompatible.

Detecting the use of a part with the wrong *semantic* interface requires an understanding of the system and some intelligence.

- To tell if a 7408 is used where a 7400 should be requires thinking about the meaning and intended function

Implementation

Sticking with the IC example:
The Interface does not describe:

- Whether it was designed with VHDL, Verilog, or etc.
- Whether the IC contains Bipolar or field effect transistors — except indirectly as this affects voltage levels, current drain, and other observable quantities.
- How many transistors it has.
- Etc.

These do not affect the suitability of the IC for its purpose and thus are not considered part of the interface.
They are considered matters of *implementation*.
Separating matters of interface from matters of implementation is a key concept in (software) engineering.
The interface abstracts what the client needs to know to use the component.

Syntax, Semantics and Implementation in Software

Syntactic interface

```
class Stack {
    public Stack()
    public boolean empty()
    public void put(int i)
    public int get() }
```

Syntactic + Semantic Interface

```
class Stack {
    /* Create a stack object, initially empty */
    public Stack()
    /* true if there is nothing in the stack, false otherwise */
    public boolean empty()
    /* Add an integer to the top of the stack */
    public void put(int i)
    /* If the stack in not empty, remove and return the integer
   most recently put on the stack, but not yet removed. */
    public int get() }
```

Syntactic interface + Implementation

```
class Stack {
    private ArrayList<Integer> contents ;
    public Stack() {
```

```
        contents = new ArrayList<Integer>() ; }
public boolean empty() {
        return contents.size() == 0 ; }
public void put(int i) {
        contents.add(i) ; }
public int get() {
        return contents.remove(contents.size() - 1 ) ; }
```

Components in Software

Various entities may serve as *components*.

- Source files
- Variables
- Statements
- Subroutines (procedures, “functions”, methods, operations)
- Classes / Types / Interfaces
- Modules / Packages / Libraries
- Subsystems
- Programs
- Resources (e.g. icons and other data)

Note components are directly created by the software designer.
(Some writers use the term “*module*” instead.)
(Some writers use the term “*component*” slightly differently.)

(UML uses the word “*component*” slightly differently.)

Components have Run-Time Instances

Compile time	Run Time
Programs	Processes
Classes	Objects
Variables	Locations
Subroutines	Invocations

Don't confuse "run-time" with "compile-time".

Components form a "containment" Hierarchy

Many components are "containers" for other components. For example:

- Programs contain subsystems or packages
- Subsystems contain packages
- Packages contain other packages, classes, types, variables, subroutines
- Classes contain variables (members), subroutines (methods), & other classes
- Subroutines contain variables and (depending on language) other subroutines

Packages exist primarily to group together related components.

Is this hierarchy a tree?

Often the same component will be included in multiple programs – sharing.

But generally speaking the hierachy is a tree.

Levels of Design

Architectural level

- the system is decomosed into subsystems and subsystems into packages
- relationships between subsystems (or between packages) are defined

Package level

- Packages are decomposed into classes.
- Relationships between classes are defined
- Class interfaces are defined.
- (UML is particularly good at this level)

Class level

- Classes are implemented with fields and subroutines

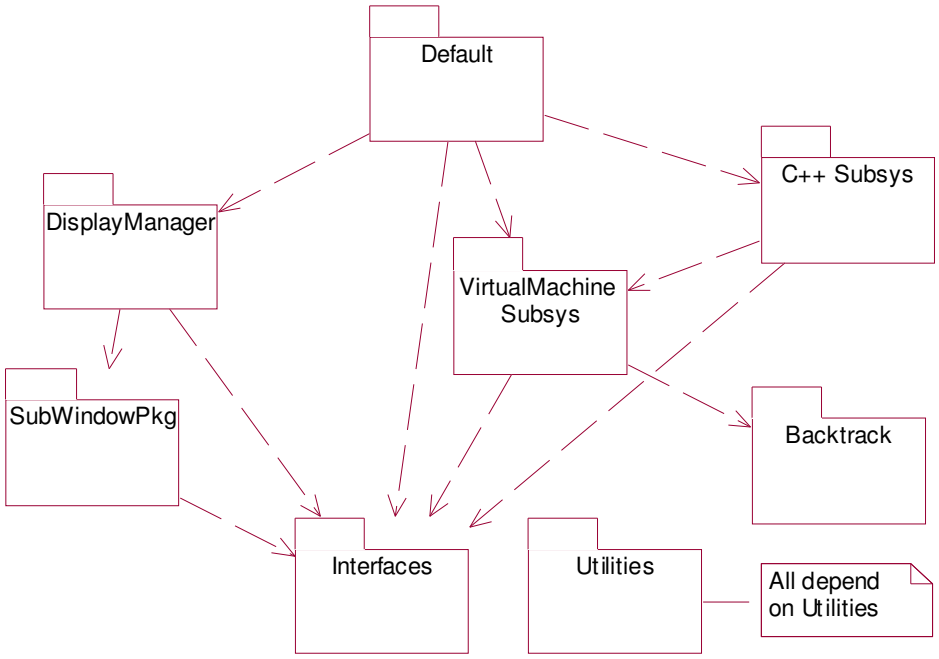
Subroutine level

- Subroutines are implemented using statements

An example Hierarchy (Next few slides)

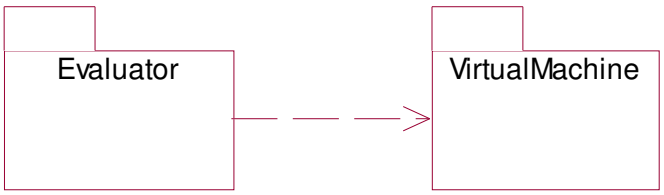
Architectural Level

Example program: The Teaching Machine TSN/MPBL. About 500 classes.



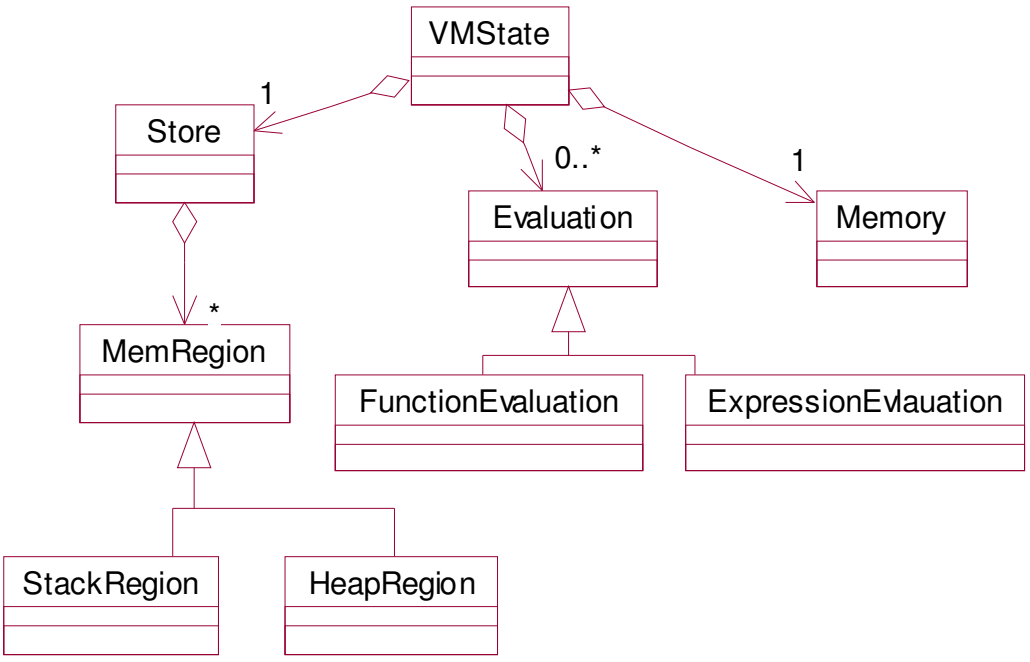
The Subsystems of the Teaching Machine.

Arrows show dependency.
A – – –> B means *A depends on B.*



The Virtual Machine Subsystem has 2 packages.

Package Level (virtualMachine package)

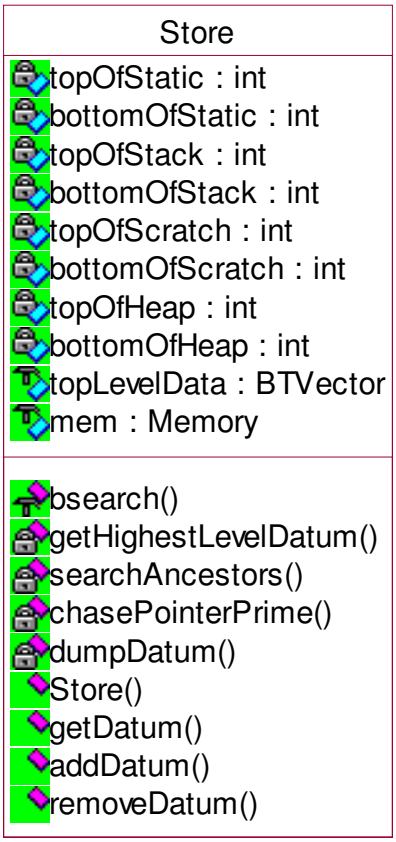


Some of the Classes within the VirtualMachine package.

C —▷ P means C *is a specialization* of P. I.e. C *inherits from* P.

W ♢—> P means every W object *has some* P objects.

Class Level ("Store" class)



showing variables (attributes / fields / data members) and subroutines (operations / methods / function members). Only the names of subroutines are shown, not the signatures.