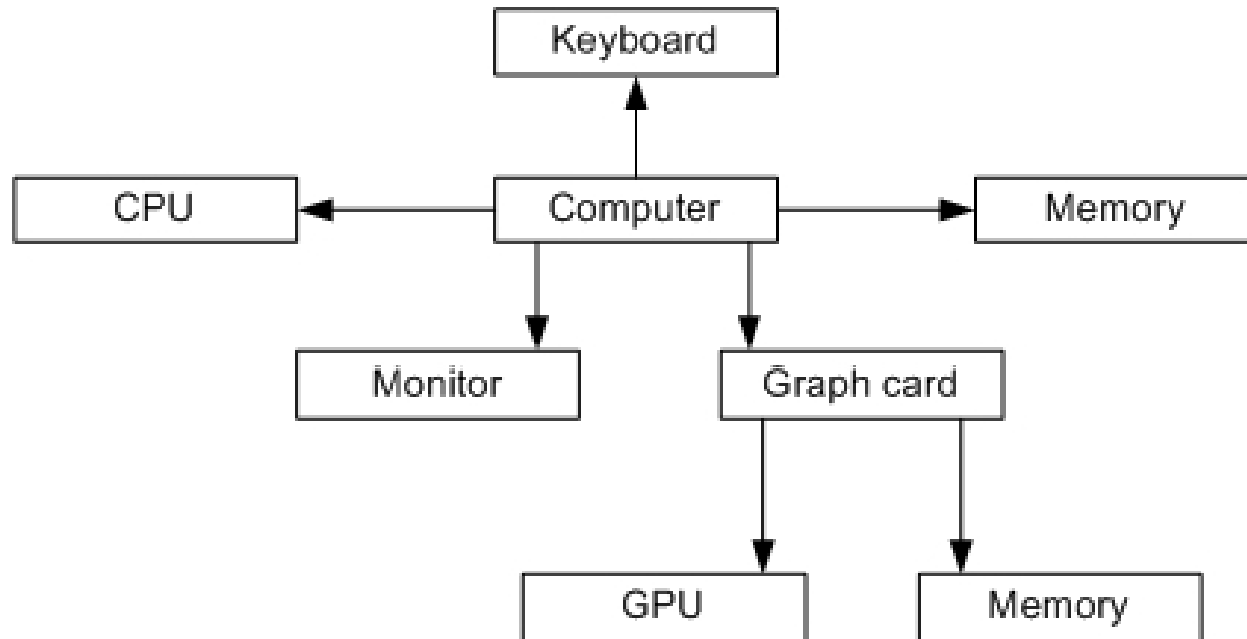


# Composite and Visitor Pattern

# Computer structure



- Every part has price
- the price of a component part is the sum of all its (recursive) sub parts

# Component

deferred class

COMPUTER\_COMPONENT

feature -- Access

*price*: INTEGER is

-- Price of this computer part

deferred

ensure

good\_result: Result >= 0

end

end

# Composite

```
class
  COMPUTER_COMPOSITE
inherit
  COMPUTER_COMPONENT

feature -- Access
  parts: LINKED_LIST [COMPUTER_COMPONENT]
    -- Parts in Current composite

  price: INTEGER is
    -- Price of this composite part
  do
    from
      parts.start
    until
      parts.after
    loop
      Result := Result + parts.item.price
      parts.forth
    end
  end
end

end
```

# CPU

class

  CPU

inherit

  COMPUTER\_COMPONENT

*redefine price* end

**feature -- Access**

*price*: INTEGER

    -- Price of this computer part

**feature - Operations**

*perform\_addition* is ...

*perform\_substraction* is ...

end

# Monitor

```
class
  MONITOR

inherit
  COMPUTER_COMPONENT
  redefine price end

feature -- Access

  price: INTEGER
    -- Price of this computer part

feature - Operations

  display is ...

end
```

# Assemble a computer

```
graph_card: COMPUTER_COMPOSITE  
main_memory, graph_memory: MEMORY
```

```
cpu: CPU  
gpu: GPU  
monitor: MONITOR  
computer: COMPUTER_COMPOSITE  
...
```

```
-- Initialize graph card.  
graph_card.parts.extend(gpu)  
graph_card.parts.extend(graph_memory)
```

```
-- Initialize computer.  
computer.parts.extend(cpu)  
computer.parts.extend(main_memory)  
computer.parts.extend(monitor)  
computer.parts.extend(graph_card)
```

```
-- Print price.  
print(computer.price)
```

What if we just want price of memories?

What if we just want price of CPU or GPU?

What if ...

# Visitor Pattern

- Process the elements of an (unbounded) data structure.
- Apply computations depending on the type of data node.
- Store the code outside of the data structure.
- Most of the time used together with the composite pattern.



# COMPUTER\_VISITOR (1/2)

deferred class

COMPUTER\_VISITOR

feature -- Process

*process\_cpu* (*a\_cpu*: CPU) is deferred end  
-- Process *a\_cpu*.

*process\_monitor* (*a\_monitor*: MONITOR) is deferred end  
-- Process *a\_monitor*.

*process\_memory* (*a\_memory*: MEMORY) is deferred end  
-- Process *a\_memory*.

*process\_gpu* (*a\_gpu*: GPU) is deferred end  
-- Process *a\_gpu*.

## COMPUTER\_VISITOR (2/2)

```
process_composite(a_composite: COMPUTER_COMPOSITE) is
  -- Process a_composite.
do
  from
    a_composite.parts.start
  until
    a_composite.parts.after
  loop
    a_composite.parts.item.process(Current)
    a_composite.parts.forth
  end
end
end

end
```

# Component

deferred class

COMPUTER\_COMPONENT

feature -- Process

*process* (*a\_visitor*: COMPUTER\_VISITOR) is  
-- Process *Current* using *a\_visitor*.

deferred

end

end

# CPU

```
class CPU  
inherit COMPUTER_COMPONENT
```

```
feature -- Access
```

```
  price: INTEGER  
    -- Price of this computer part
```

```
feature - Operations
```

```
  perform_addition is ...  
  perform_substraction is ...
```

```
feature -- Process
```

```
  process(a_visitor: COMPUTER_VISITOR) is  
    -- Process Current using a_visitor.  
  do  
    a_visitor.process_cpu(Current)  
  end
```

```
end
```

# MEMORY

```
class MEMORY
inherit COMPUTER_COMPONENT

feature -- Access

    price: INTEGER
        -- Price of this computer part

feature - Operations
    display is ...

feature -- Process

    process(a_visitor: COMPUTER_VISITOR) is
        -- Process Current using a_visitor.
    do
        a_visitor.process_memory(Current)
    end

end
```

# Composite

class

COMPUTER\_COMPOSITE

inherit

COMPUTER\_COMPONENT

feature -- Access

*parts*: LINKED\_LIST [COMPUTER\_COMPONENT]

-- Parts in Current composite

feature -- Process

*process* (*a\_visitor*: COMPUTER\_VISITOR) is

-- Process *Current* using *a\_visitor*.

do

*a\_visitor.process\_composite* (Current)

end

end

# MEMORY\_PRICE\_VISITOR (1/2)

class

MEMORY\_PRICE\_VISITOR

inherit

COMPUTER\_VISITOR

feature -- Calculate

*memory\_price* (*a\_component*: COMPUTER\_COMPONENT): INTEGER is

-- Price of all kinds of memory in *a\_component*

do

*last\_price* := 0

*a\_component.process* (Current)

end

feature -- Access

*last\_price*: INTEGER

-- Last calculated price

# MEMORY\_PRICE\_VISITOR (2/2)

**feature** -- Process

```
process_memory (a_memory: MEMORY) is  
  do  
    last_price := last_price + a_memory.price  
  end
```

```
process_cpu (a_cpu: CPU) is  
  do  
  end
```

```
process_monitor (a_monitor: MONITOR) is  
  do  
  end
```

```
process_gpu (a_gpu: GPU) is  
  do  
  end
```

**end**



# Double Dispatch

*component := memory*

*visitor := memory\_price\_visitor*

...

*component.process(visitor)*