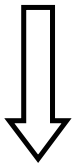
vysoká kohézia a nízka zviazanosť



modulárna zrozumiteľnosť,
modulárna spojitosť,
modulárna ochrana,
modulárna komponovateľnosť

správnosť, robustnosť,
flexibilnosť, znovupoužiteľnosť

```
template <class G> class Stack
{
public:
        void put (G element);
        void remove ();
        G item ();
        bool empty ();
private:
        G *array;
        int count;
}
```

TYPES
- **STACK [G]**

FUNCTIONS
- **put: STACK [G] x G → STACK [G]**
- **remove: STACK [G] → STACK [G]**
- **item: STACK [G] → G**
- **empty: STACK [G] → BOOLEAN**
- **new: STACK [G]**

AXIOMS
x: G, s: STACK [G]
- **item (put (s, x)) = x**
- **remove (put (s, x)) = s**
- **empty (new) = true**
- **not empty (put (s, x))**

PRECONDITIONS
- **remove (s: STACK [G]) require not empty (s)**
- **item (s: STACK [G]) require not empty (s)**

remove **is**
	-- Remove top element.
require
	not *empty*
do
	...
end

*put (*x: G) **is**
	-- Add *x* on top.
do
	...
ensure
	not *empty*
	item = x
end

TYPES
- **STACK2 [G]**

FUNCTIONS
- **put: STACK2 [G] x G → STACK2 [G]**
- **remove: STACK2 [G] → STACK2 [G]**
- **make: INTEGER → STACK2 [G]**
- **item: STACK2 [G] → G**
- **capacity: STACK2 [G] → INTEGER**
- **count: STACK2 [G] → INTEGER**
- **empty: STACK2 [G] → BOOLEAN**
- **full: STACK2 [G] → BOOLEAN**

AXIOMS
x: G, s: STACK2 [G], n: INTEGER
- **item (put (s, x)) = x**
- **remove (put (s, x)) = s**
- **capacity (make (n)) = n**
- **capacity (put (s, x)) = capacity (s)**
- **count (make (n)) = 0**
- **count (put (s, x)) = count (s) + 1**
- **empty (s) = (count (s) = 0)**
- **full (s) = (count (s) = capacity (s))**

PRECONDITIONS
- **put (s: STACK2 [G], x: G) require not full (s)**
- **remove (s: STACK2 [G]) require not empty (s)**
- **item (s: STACK2 [G]) require not empty (s)**
- **make (n: INTEGER) require n >= 0**

```
indexing
        description: "Stacks: Dispenser structures with a Last-In, First-Out
                        access policy, and a fixed maximum capacity"

class STACK2 [G] creation
        make

feature – Initialization

        make (n: INTEGER) is
                        -- Allocate stack for a maximum of n elements
        require
                positive_capacity: n >= 0
        do
                capacity := n
                !! representation.make (1, capacity)
        ensure
                capacity_set: capacity = n
                array_allocated: representation /= Void
                stack_empty: empty
        end

feature -- Access

        capacity: INTEGER
                        -- Maximum number of stack elements

        count: INTEGER
                        -- Number of stack elements

        item: G is
                        -- Top element
        require
                not_empty: not empty    -- i.e. count > 0
        do
                Result := representation [count]
        end

feature -- Status report

        empty: BOOLEAN is
                        -- Is stack empty?
        do
                Result := (count = 0)
        ensure
                empty_definition: Result = (count = 0)
        end

        full: BOOLEAN is
                -- Is stack full ?
        do
                Result := (count = capacity)
        ensure
                full_definition: Result = (count = capacity)
        end

feature -- Element change

        put (x: G) is
                        -- Add x on top
```

```
        require
                not_full: not full
                        -- i.e. count < capacity
        do
                count := count + 1
                representation.put (count, x)
        ensure
                not_empty: not empty
                added_to_top: item = x
                one_more_item: count = old count + 1
                in_top_array_entry: representation [count] = x
        end

        remove is
                        -- Remove top element
        require
                not_empty: not empty -- i.e. count > 0
        do
                count := count – 1
        ensure
                not_full: not full
                one_fewer: count = old count – 1
        end

feature {NONE} -- Implementation

        representation: ARRAY [G]
                        -- The array used to hold the stack elements

invariant
        count_non_negative: 0 <= count
        count_bounded: count <= capacity
        consistent_with_array_size: capacity =
                                                representation.capacity
        empty_if_no_elements: empty = (count = 0)
        item_at_top: (count > 0) implies
                                        (representation [count] = item)
end -- class STACK2



if x < 0 then
        „Spracuj chybu, nejako“
else
        „Vypocitaj sqrt“
end


x = a*a + b*b;
...
assert (x >= 0);
        // lebo x bolo vypočítané ako súčet štvorcov
y = sqrt (x);
```

```
status = commit_transaction (&t);
if (status == CONSTRAINT_VIOLATION)
{
        ...
} else if (status == TIMED_OUT)
{
        ...
} else if (status == NOT_ENOUGH_MEMORY)
{
        ...
} else if (status == CONNECTION_TO_DB_LOST)
{
        ...
}

try
{
        commit_transaction (&t);
}
catch (...)
{
        ...
}


my_document.print (printer_name, paper_size, color_or_not, postscript_level, print_resolution)


mydoc.set_printer_name ("LPA0:");
mydoc.set_color ();
mydoc.set_resolution (600);
mydoc.print ();


mydoc.print_with_size_and_resolution ("A4", 300);

const CString& CString::operator=(LPCTSTR lpsz)
{
        ASSERT(lpsz == NULL || AfxIsValidString(lpsz));
        AssignCopy(SafeStrlen(lpsz), lpsz);
        return *this;
}
```

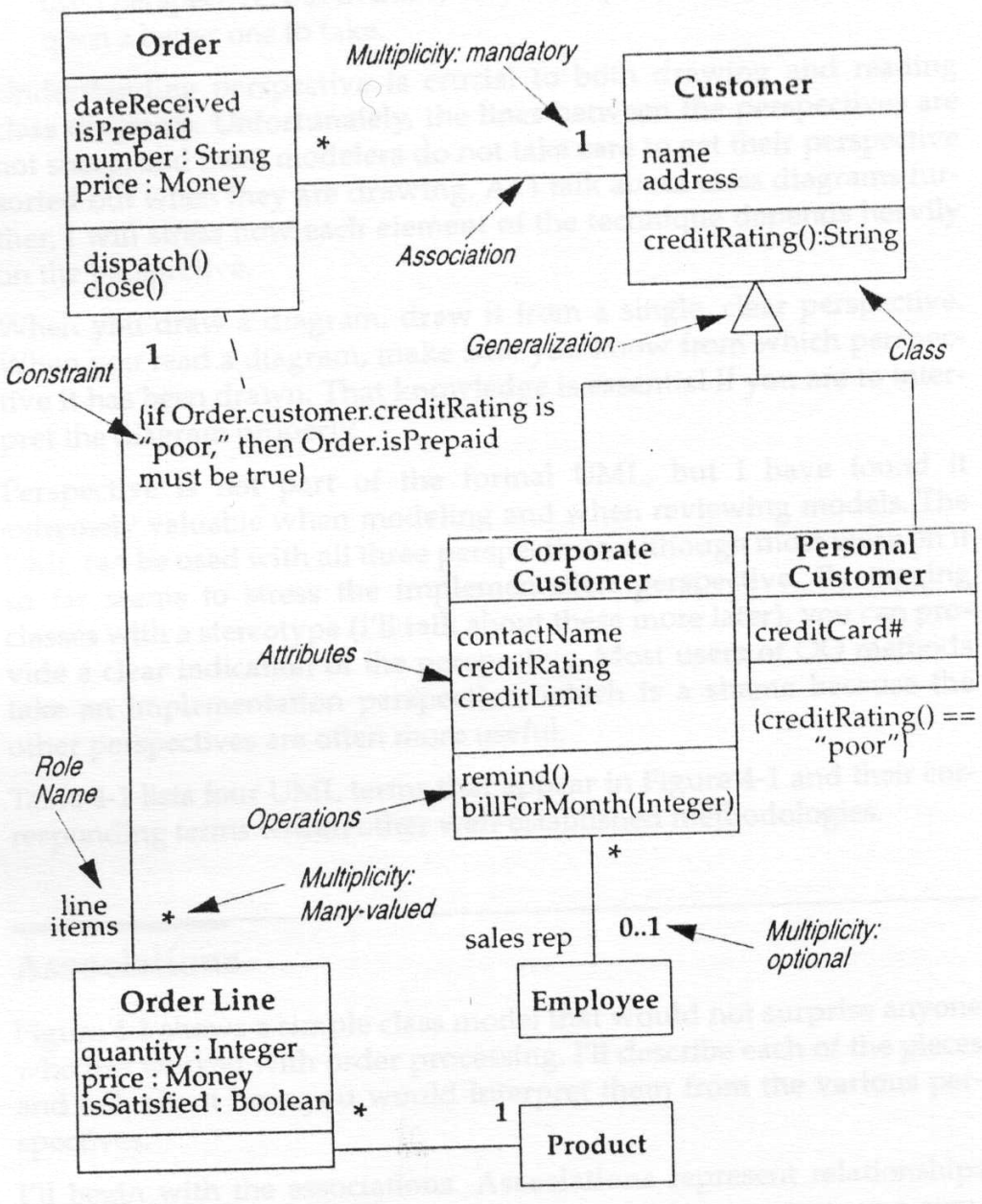
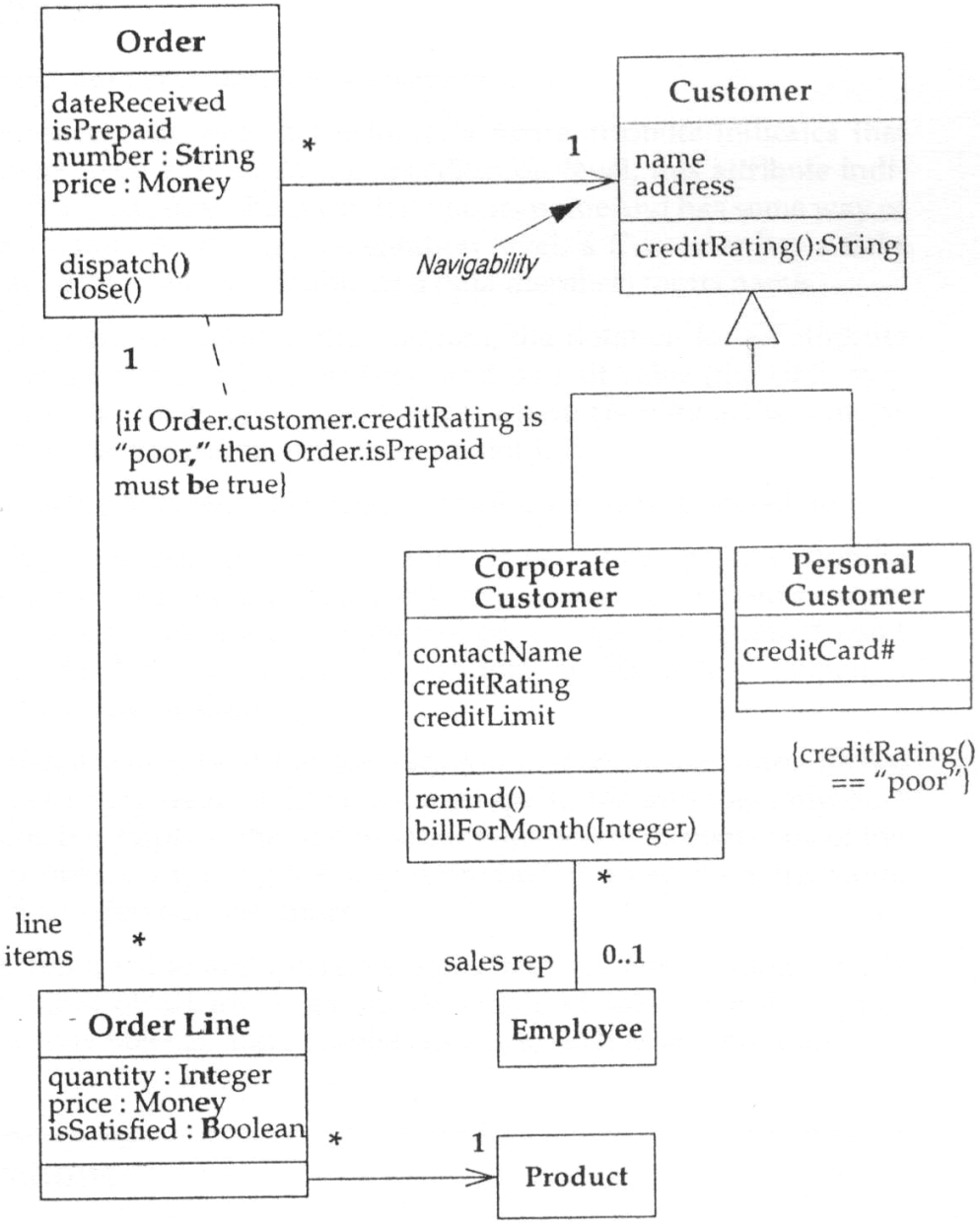


Figure 4-1: *Class Diagram*


Order
dateReceived
isPrepaid
number : String
price : Money
dispatch()
close()
*
1
Customer
name
address
creditRating():String
Navigability
1
{if Order.customer.creditRating is "poor," then Order.isPrepaid must be true}
Corporate Customer
contactName
creditRating
creditLimit
remind()
billForMonth(Integer)
Personal Customer
creditCard#
{creditRating() == "poor"}
*
line items
*
sales rep
0..1
Order Line
quantity : Integer
price : Money
isSatisfied : Boolean
Employee
*
1
Product