Exercise Session Informatik III

9. Introduction to Eiffel

A Simple Example

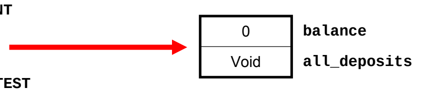
```
indexing
  description: "Simple Bank Accounts"
class
  ACCOUNT
feature -- Access
  balance: INTEGER
              -- Current Balance
  deposit_count: INTEGER is
              -- Number of deposits made since opening
    do
      if all_deposits /= Void then
        Results := all_deposits.count
      end
    end
```

A Simple Example

```
feature -- Element change
  deposit(sum: INTEGER) is
              -- Add sum to account
    do
      if all_deposits = Void then
        create all_deposits
      end
      all_deposits.extend(sum)
      balance := balance + sum
    end
feature -- Implementation
  all_deposits: DEPOSIT_LIST
              -- List of deposits since account opening
end -- class ACCOUNT
```

Object Creation

```
class TEST
feature
  do_test is -- Simple test routine
    local
      x: ACCOUNT
    do
      create x
    end
end -- class TEST
```



Type	Default Value
INTEGER, REAL, DOUBLE	Zero
BOOLEAN	False
CHARACTER	Null
Reference Types	Void Reference
Composite Expanded Types	Same rules, applied recursively to all fields

Object Creation Routines

```
indexing
  description: "Simple accounts, with first deposit"
class
  ACCOUNT_WITH_DEPOSIT
create
  make
feature -- Initialisation
  make(sum: INTEGER) is
        -- Initialise account with sum
    do
      deposit(sum)
    end
...
end -- class ACCOUNT_WITH_DEPOSIT
```

Client:
`create x.make(2000)`

Expanded Classes

- It is possible to declare an entire class as expanded. This means that a variable of this type will always contain the object itself and not a reference to the object.

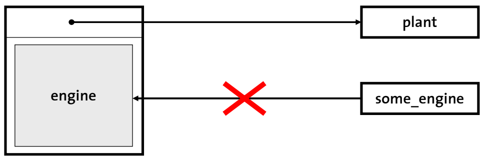
```
indexing
  description: "Integer Values"
expanded class
  INTEGER
feature -- Basic Operations
  infix "+" (other: INTEGER): INTEGER is
    do ... end
...
end -- class INTEGER
```

- On the other hand, one can achieve this behavior for only one variable of a not expanded type, using the following syntax:

```
x: expanded ACCOUNT
```

Expanded Classes

- As a consequence of expanded classes or variables, sharing of these object is no longer possible!



```
class CAR feature
  originating_plant: PLANT
  engine: expanded ENGINE
end -- class CAR
```

Access Control

- Sometimes it makes sense that a client of a class cannot access all features of a class.
- Eiffel provides a powerful mechanism to specify who can access a feature
  - everybody — `feature` or `feature {ANY}`
  - special classes — `feature {ACCOUNT, LIST}`
  - nobody — `feature {NONE}` or `feature { }`
- Attributes are always read-only in Eiffel. To update or set an attribute a routine has to be written

```
feature {...} set_attribute(v: VALUE_TYPE) is
  do
    attribute := v
  end
```

That's all folks!

Bis nächste Woche:
• EiffelStudio runterladen (www.eiffel.com)
• Einarbeiten in EiffelStudio mit Tutorial