OOABL - Design Pattern

Presentation of common OO Design Pattern in OO ABL environment.

Design Pattern

- „In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.“ -Wikipedia
- Design Patterns: Elements of Reusable Object-Oriented Software
- Three Types:
  - Creational Pattern
  - Struktural Pattern
  - Behavioral Pattern
- 23 GoF Pattern
- Seven pattern of practical use will be discussed today

Pattern 1: Builder

- Type: Creational Pattern
- Use one object to prepare the creation of another object
- Use if the constructor has a lot of parameter
- Why use it?
  - More readable
  - Parameter are type save and named
  - Auto-Complete
  - Simple add parameter later

Pattern 1: Builder

Initial situation

User
- cFirstName: CHARACTER - cLastName: CHARACTER - iAge: INTEGER - cPhone: CHARACTER - cAddress: CHARACTER
+ User(cFirstName: CHARACTER, cLastName: CHARACTER) + User(cFirstName: CHARACTER, cLastName: CHARACTER, iAge: INTEGER) + User(cFirstName: CHARACTER, cLastName: CHARACTER, iAge: INTEGER, cPhone: CHARACTER) + User(cFirstName: CHARACTER, cLastName: CHARACTER, iAge: INTEGER, cPhone: CHARACTER, cAddress: CHARACTER)

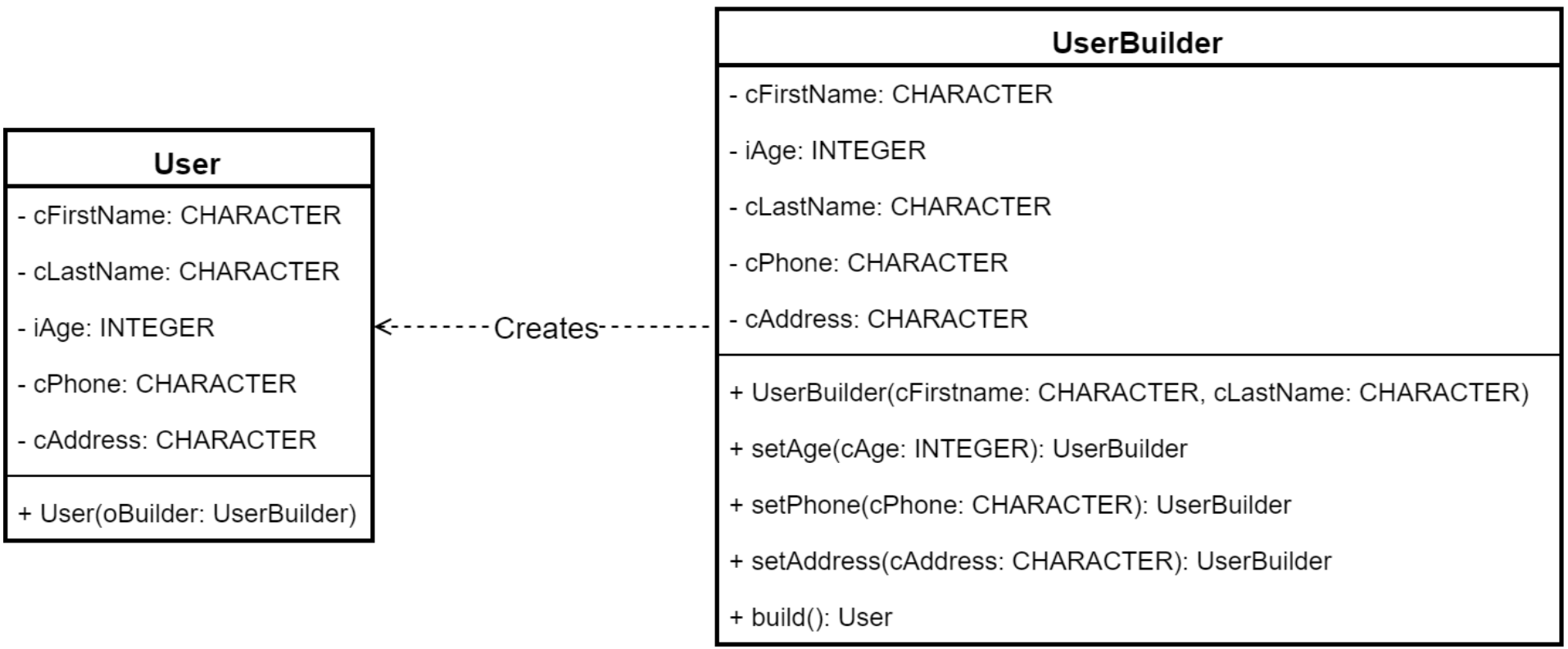
Pattern 1: Builder

Initial code with a lot of parameters:

```
DEFINE VARIABLE oUser AS User NO-UNDO.

oUser = NEW User(
  "Michael",
  "Barfs",
  23,
  "+49 40-30 68 03-26",
  "Valentinskamp 30, 20355 Hamburg"
).
```

Pattern 1: Builder

With Builder Pattern:



Pattern 1: Builder

Builder – part of a setter:

```
CLASS UserBuilder:
…
  METHOD PUBLIC UserBuilder setAge(iAge AS INTEGER):
    THIS-OBJECT:iAge = iAge.
    RETURN THIS-OBJECT.
  END METHOD.
…
END CLASS.
```

Pattern 1: Builder

Builder call:

```
DEFINE VARIABLE oUser AS User NO-UNDO.
oUser =
  (NEW UserBuilder("Michael", "Barfs")
  :setAge(23)
  :setPhone("+49 40-30 68 03-26")
  :setAddress("Valentinskamp 30, 20355 Hamburg")
  :build()).
```

Pattern 1: Builder

Small part:

```
RUN StatusCreate IN l-Import-Library-
Handle
   ( INPUT    l-DB-Cust,

     INPUT    "",

     INPUT    150,
...
     INPUT
       "QtyType=" + OrderQtyQualifier
     + "{&T}"
       + "UTCTime=" + l-UTCTime
       + "{&T}"
       + "ConC-ID=" + SSCO-Ord.ConC-ID
...
) NO-ERROR.
```

Sample: extreme # of parameter:

```
RUN StatusCreate IN l-Import-Library-Handle
  ( INPUT   l-DB-Cust,                  /* Cust Code */
    INPUT   "",                         /* Cnee Code */
    INPUT   150,                        /* status numeric */
  /* tb, 100304; export 8645 with O-E instead of O-I */
  &IF ("{&Exp_8645_with_O-E_v1}") = "TRUE" &THEN
    INPUT   "CreateNewRep2" + SSCO-Ord.OrderType + ",StartOrderExport665",  /* Create report flag */
  &ELSE
    INPUT   "CreateNewRep" + SSCO-Ord.OrderType + ",StartOrderExport665",  /* Create report flag */
  &ENDIF
    INPUT   TRUE,                       /* Report NEW = YES */
    INPUT   SSCO-o-Movement.Movement-ID, /* NOT Ord-ID */
    INPUT   "O",                        /* Status Type */
    INPUT   0,                          /* Suborder Number */
    INPUT   0,                          /* ? */
    INPUT   l-StatusDate,               /* Status Date */
    INPUT   l-StatusTime,               /* Status Time */
  /* tb, 050801 */
    INPUT   "Customer EDI",          /* User Code */
    INPUT   FALSE,                      /* Print 1 */
    INPUT   FALSE,                      /* Print 2 */
    INPUT   ?,                          /* default is Today */
    INPUT   "",                         /* Remarks */
    INPUT   SSCO-Ord.OrdQty,            /* Qty */
    INPUT   0,                          /* info code */
    INPUT   SSCO-Ord.Send-ID,           /* Send-ID */
    INPUT   SSCO-Ord.Send-Code,         /* Send-Code */
    /* no transmission to CIEL for Road orderlines */
    &IF ("{&Road_Order}") = "TRUE" &THEN
    INPUT   (SSCO-Ord.TrnsType-Code <> "R" AND b-Cust.Released), /* IsTransmit */
    &ELSE
    INPUT   b-Cust.Released,            /* IsTransmit */
    &ENDIF
    INPUT   l-Import-Date-asDate,       /* created on */
    INPUT   l-Import-Time-asChar,       /* time on */
    INPUT   "",                         /* knref */
    INPUT   "",                         /* damaged code */
    INPUT   "",                         /* address type-code */
    INPUT   ?,                          /* docs delivery date */
    INPUT   "",                         /* docs delivery time */
    INPUT   0,                          /* invoice header ID */
    INPUT   TRUE,                       /* check for duplicate status ? */
    INPUT   "",                         /* Reason Code */
    INPUT   "",                         /* Export/Import Flag */
    INPUT   "",                         /* SubStatus */
    INPUT   "QtyType=" + l-tt-{&ShipType}660.OrderQtyQualifier       + "{&T}" +
            "UTCTime=" + l-UTCTime                                   + "{&T}" +
            "ConC-ID=" + STRING(SSCO-Ord.ConC-ID), /* additional Fields ({&T}-separated list */
    OUTPUT  l-Stat-Code,                /* status code. if ? then status invalid */
    OUTPUT  l-Return-Code               /* returncode passed by called procedure */
) NO-ERROR.
```

Pattern 1: Builder

This call with Builder (part of):

```
DEFINE VARIABLE oStatusCreate AS StatusCreate NO-UNDO.

oStatusCreate =
  (NEW StatusCreateBuilder()
  :setCustCode(l-DB-Cust)
  :setStatusNumeric(150)
...
  :setQtyType(OrderQtyQualifier)
  :setUTCTime(l-UTCTime)
  :setConCID(SSCO-Ord.ConC-ID)
...
  :build()).
```

Builder - Discussion

- Advantages
  - Improves readability
  - Named parameters
  - Auto-Complete supported
  - Allows late changes
- Practical use in 4 GL
  - Very good
- Disadvantages
  - 'None'
    (Multiple calls need time)
- Pattern or Anti-Pattern
  - Pattern
    - Avoid hidden validations
    - Avoid nesting objects
    - Avoid hierarchical structures
      (call is linear)

Pattern 2: Singleton

- Type: Creational Pattern
- Kind of "global objects" in OO
- When to use
  - Need a global, single object all over the application
- Why to use:
  - Inheritance possible
  - Has some logic during instantiation
  - Saves resources
- Examples:
  - Configuration
  - Communication setup

Pattern 2: Singleton

Class with Singleton Pattern:

```
CLASS Konfiguration:
...
  DEFINE PUBLIC STATIC PROPERTY oInstance AS Configuration
    PUBLIC GET():
      IF oInstance = ? THEN
        oInstance = NEW Configuration().
      RETURN oInstance.
    END GET.
    PRIVATE SET.

  CONSTRUCTOR PRIVATE Configuration():
    loadConfig().
  END CONSTRUCTOR.
...
END CLASS.
```

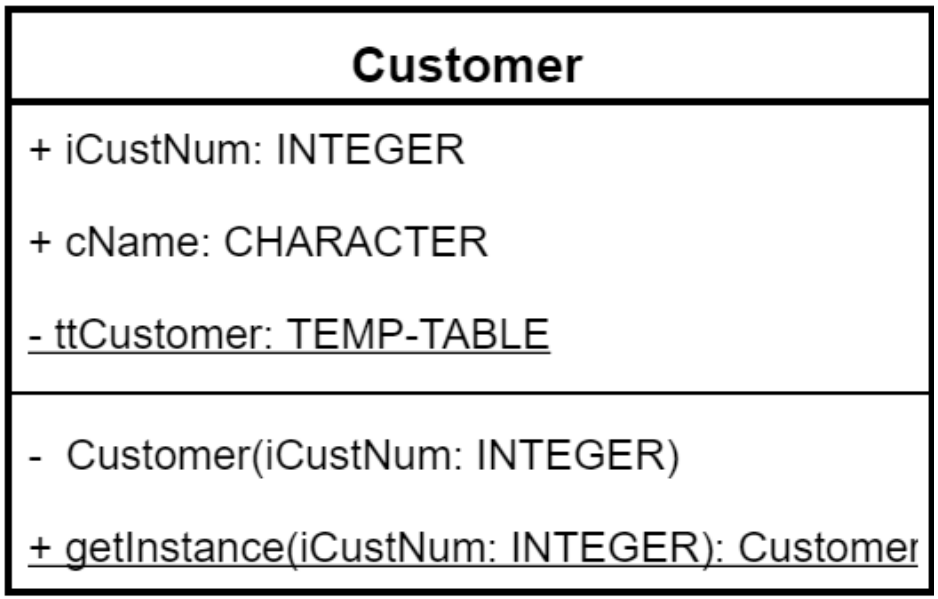
Pattern 2: Singleton

Singleton call:

```
DEFINE VARIABLE oKonf AS Configuration NO-UNDO.

oKonf = Configuration:oInstance.

oKonf:setValue("mode", "debug").
oKonf:saveToFile().
```

Pattern 3: Multiton

- Type: Creational Pattern
- One static access method
- Objects saved with ID
- When to use:
  - N objects (data members) will be accessed randomly
- Why to use:
  - Performance
  - Save ressources
  - Simple code



Pattern 3: Multiton

Sample part 1 (static Temp-Table):

```
...
  DEFINE PUBLIC PROPERTY iCustNum AS INTEGER NO-UNDO GET. PRIVATE SET.
  DEFINE PUBLIC PROPERTY cName AS CHARACTER NO-UNDO GET. PRIVATE SET.

  DEFINE PRIVATE STATIC TEMP-TABLE ttCustomer
    FIELD custNum AS INTEGER
    FIELD obj     AS Progress.Lang.Object
    INDEX ID custNum.
  .
...
END CLASS.
```

Pattern 3: Multiton

Sample part 2 (static access method):

```
CLASS Customer:
...
  METHOD PUBLIC STATIC Customer getInstance(iCustNum AS INTEGER):
    FIND FIRST ttCustomer WHERE ttCustomer.custNum = iCustNum NO-LOCK NO-ERROR.
    IF NOT AVAILABLE ttCustomer THEN DO:
      CREATE ttCustomer.
      ASSIGN
        ttCustomer.custNum   = iCustNum
        ttCustomer.obj  = NEW Customer(iCustNum)
      .
    END.

    RETURN CAST(ttCustomer.obj, Customer).
  END METHOD.
...
END CLASS.
```

Pattern 3: Multiton

Sample part 3 (private constructor):

```
CLASS Customer:
...
  CONSTRUCTOR PRIVATE Customer(iCustNum AS INTEGER):
    DEFINE BUFFER bCustomer FOR Customer.

    FIND FIRST bCustomer WHERE bCustomer.CustNum = iCustNum NO-LOCK NO-ERROR.
    IF AVAILABLE bCustomer THEN DO:
      THIS-OBJECT:cName    = bCustomer.Name.
      THIS-OBJECT:iCustNum = bCustomer.CustNum.
    END.
  END CONSTRUCTOR.
...
END CLASS.
```

Pattern 3: Multiton

Sample part 4 (usage):

```
DEFINE VARIABLE oCust AS Customer NO-UNDO.

oCust = multiton.Customer:getInstance(1537).
```

Pattern 4: Lazy Loading

- Type: Creational Pattern
- Delay until access:
  - Object creation
  - Calculations, summaries…
  - Other expensive processing
- When to use:
  - Initialising of a resource (class, tab, communication…) takes long
- Why to use:
  - Fast start
  - Save effort for things nobody will use

Pattern 4: Lazy Loading

Sample part 1 (property):

```
CLASS Invoice:
...
  DEFINE PUBLIC PROPERTY cCustomerName AS CHARACTER NO-UNDO INITIAL ? PRIVATE SET.
    PUBLIC GET:
      IF cCustomerName = ? THEN DO:
        DEFINE VARIABLE iCN AS INTEGER NO-UNDO.
        iCN = THIS-OBJECT:iCustNum.
        DEFINE BUFFER bCustomer FOR Customer.
        FIND FIRST bCustomer WHERE bCustomer.CustNum = iCN NO-LOCK NO-ERROR.
        IF AVAILABLE bCustomer THEN DO:
          cCustomerName = bCustomer.Name.
        END.
      END.

      RETURN cCustomerName.
    END GET.
...
END CLASS.
```

Pattern 4: Lazy Loading

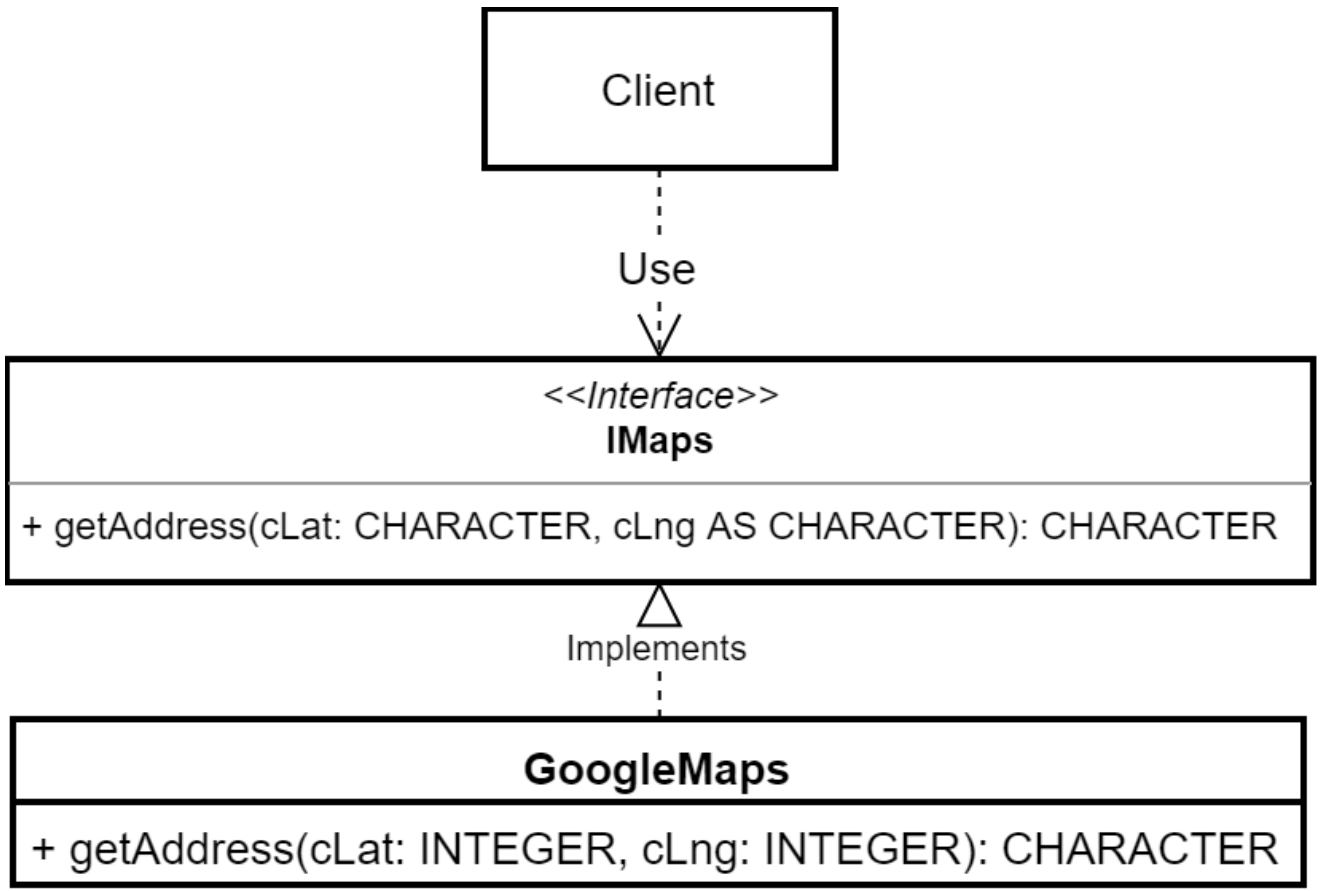
Sample part 2 (constructor & other properties):

```
CLASS Invoice:
...
  CONSTRUCTOR PUBLIC Invoice(iInvoiceNum AS INTEGER):
    DEFINE BUFFER bInvoice FOR Invoice.

    FIND FIRST bInvoice WHERE bInvoice.Invoicenum = iInvoiceNum NO-LOCK
NO-ERROR.
    IF AVAILABLE bInvoice THEN DO:
      THIS-OBJECT:iInvoiceNum = iInvoiceNum.
      THIS-OBJECT:iCustNum    = bInvoice.CustNum.
    END.
  END CONSTRUCTOR.
...
END CLASS.
```

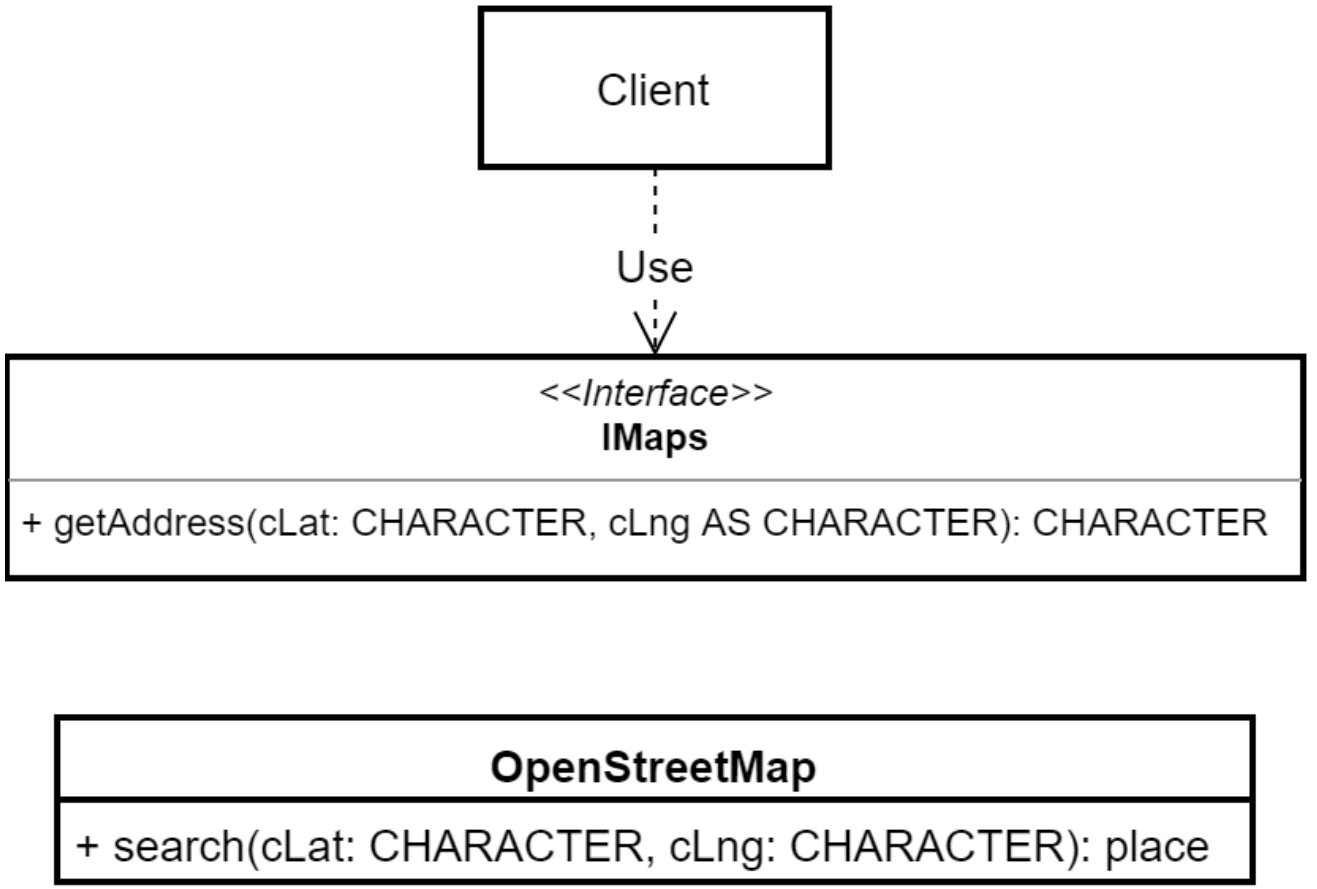
Pattern 5: Adapter

- Type: Struktural Pattern
- Combine two incompatible interfaces
- When to us:
  - Make systems more flexible
  - Wrap 3rd party / old code
- Why to use:
  - Have only one (simpler) interface
  - Integrate other libraries / 3rd party

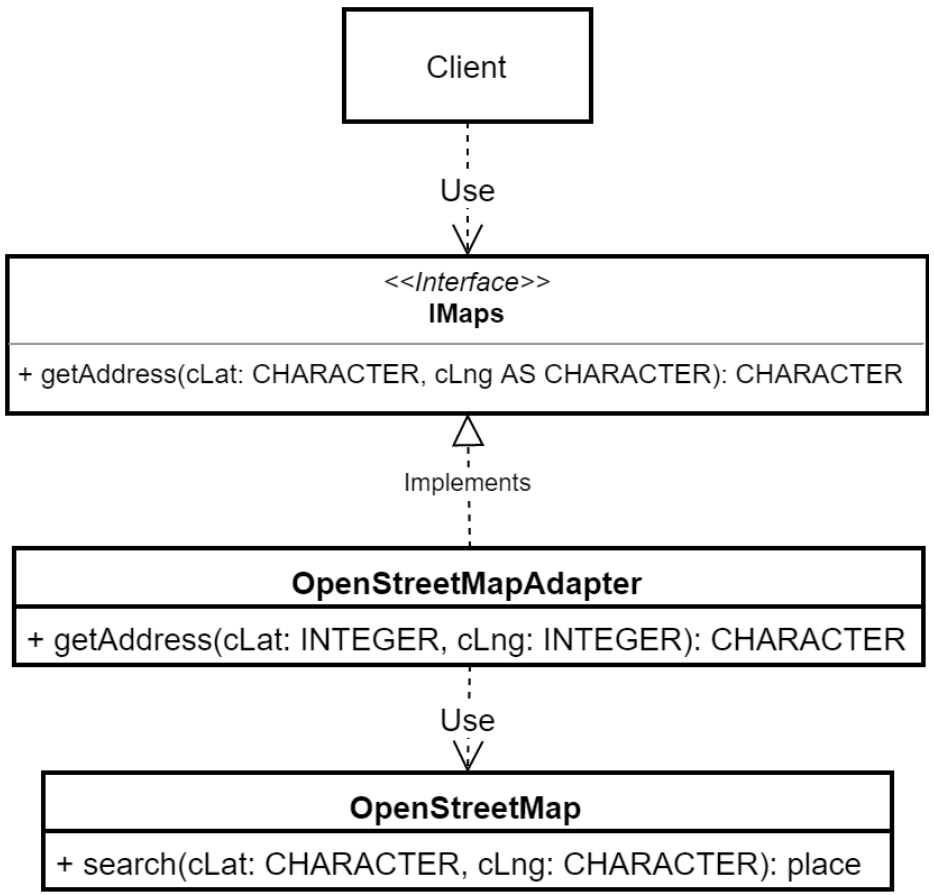
Pattern 5: Adapter



Pattern 5: Adapter



Pattern 5: Adapter



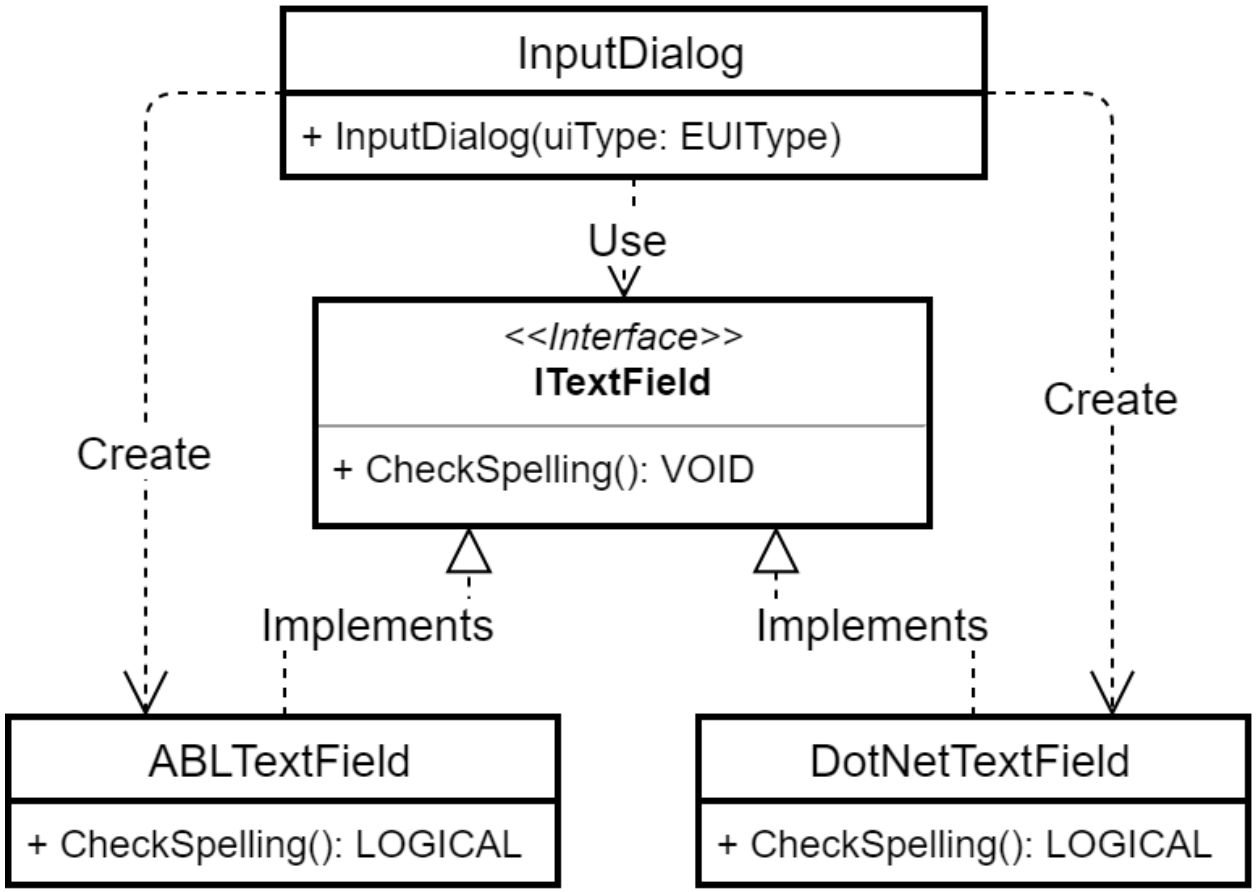
Pattern 5: Adapter

```
CLASS OpenStreetMapAdapter IMPLEMENTS IMaps:
  DEFINE PRIVATE PROPERTY oOpenStreetMap AS OpenStreetMap NO-UNDO
    PRIVATE GET.
    PRIVATE SET.

  CONSTRUCTOR PUBLIC OpenStreetMapAdapter():
    oOpenStreetMap = NEW OpenStreetMap().
  END CONSTRUCTOR.


  METHOD PUBLIC CHARACTER getAddress(cLat AS CHARACTER ,cLng AS
CHARACTER):
     RETURN oOpenStreetMap:search(cLat, cLng):Address.
  END METHOD.
END CLASS.
```

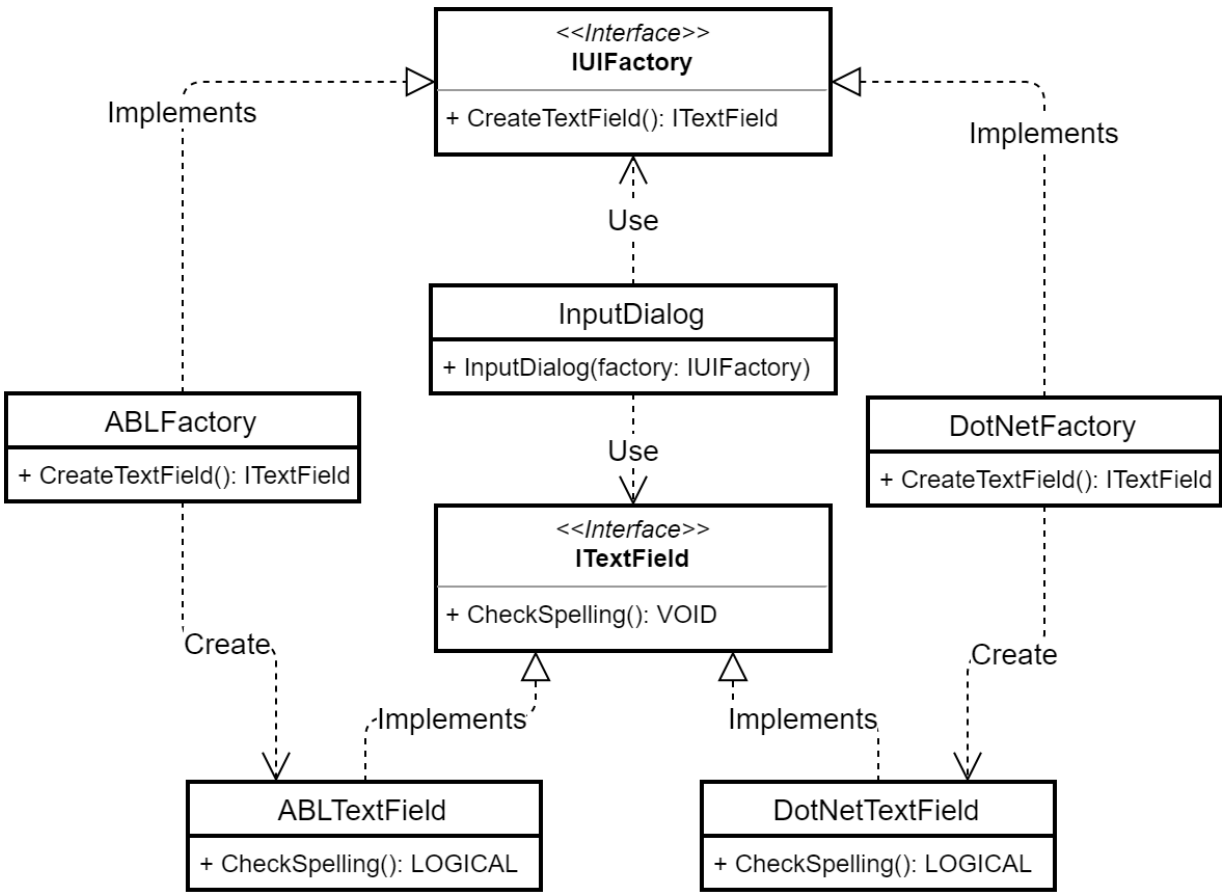
Pattern 6: Abstract Factory

- Type: Creational Pattern
- Use an abstract method for object creation
- When to use:
  - Make code more flexible
  - During compile the final class is unknown
- Why to use:
  - Have generic Interface
  - Loose coupling
  - Extensible structure
- Use samples:
  - Create UI elements (classic OE UI, .NET UI)
  - Unit testing

Pattern 6: Abstract Factory



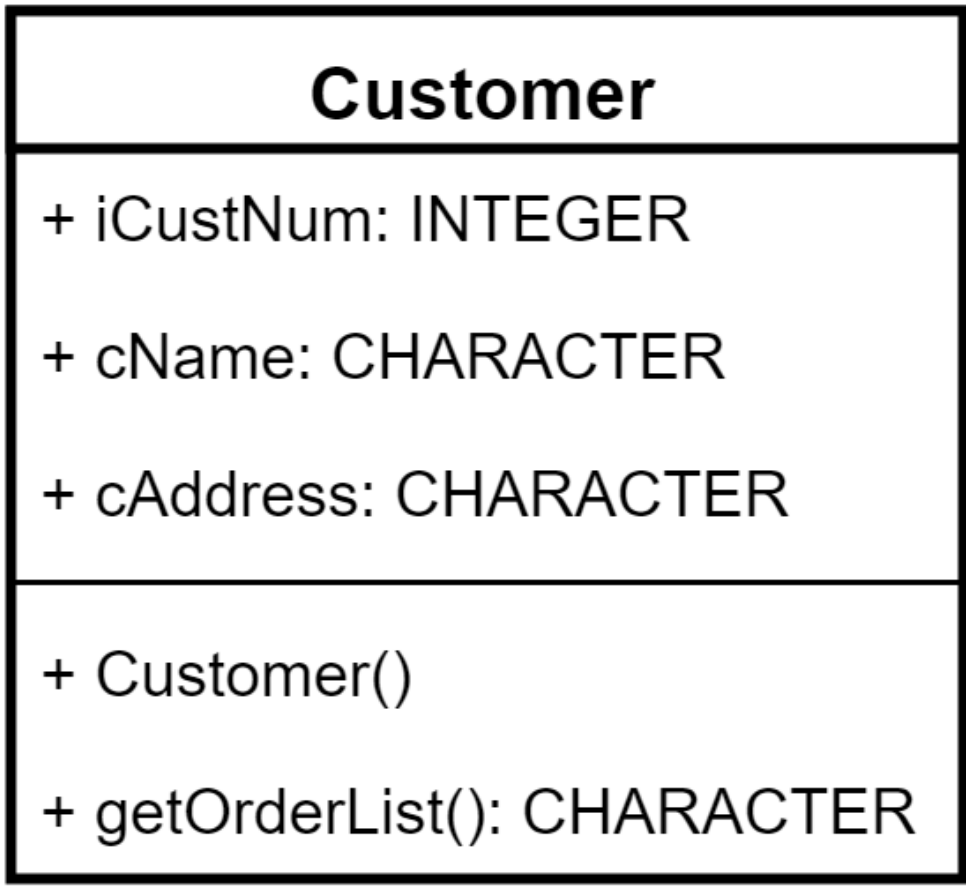
Pattern 6: Abstract Factory



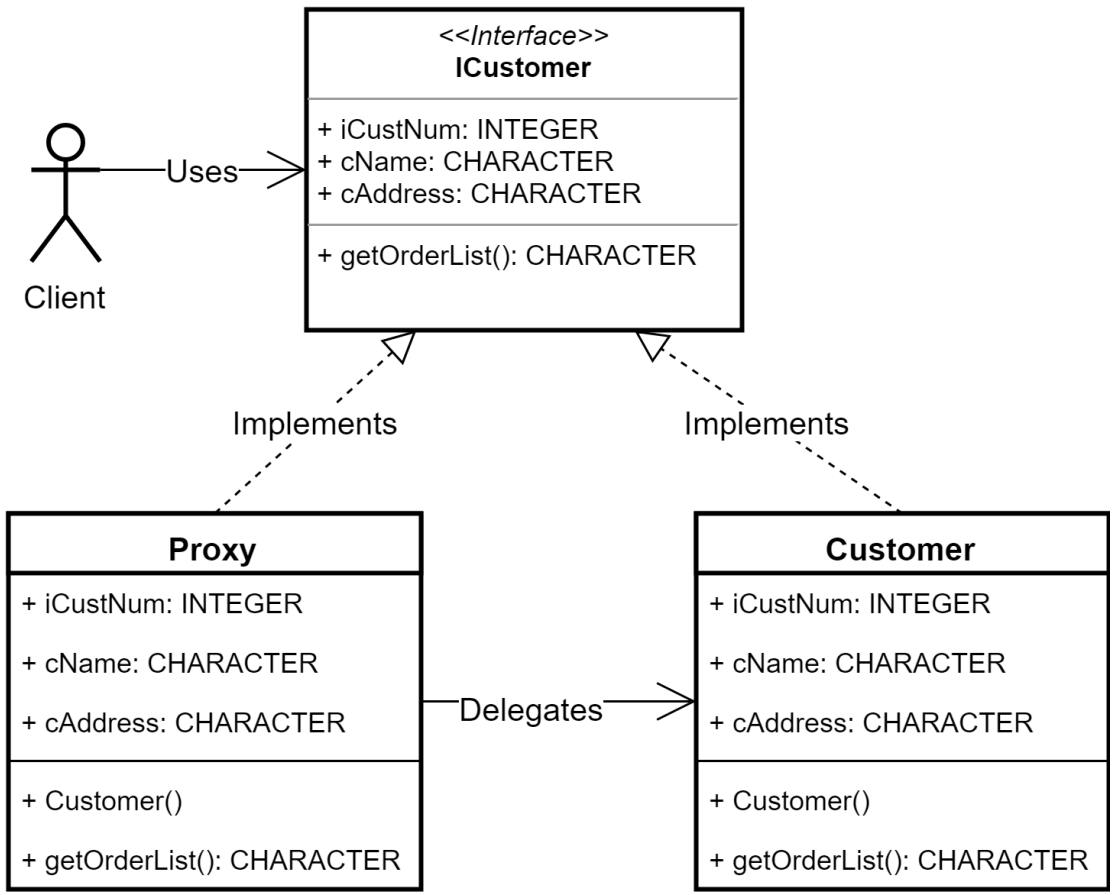
Pattern 7: Proxy

- Type: Behavioural Pattern
- Why to use:
  - Use remote objects like local objects
  - Protect an object (security)
  - Reduce visible object complexity
- Why to use:
  - More independence (interfaces)
  - Create distributed systems
  - Simpler programming
- Beispiel:
  - Authentication
  - Remote method invocation

Pattern 7: Proxy



Pattern 7: Proxy



Abstract

- Seven Pattern discussed:
  - Builder
  - Singleton
  - Multiton
  - Lazy Loading
  - Adapter
  - Abstrakte Fabrik
  - Proxy
- A company should defines pattern policies
- When there is a useful pattern, use it