Signsoft

Einsatz von Jakarta Commons

Jennifer Wagner

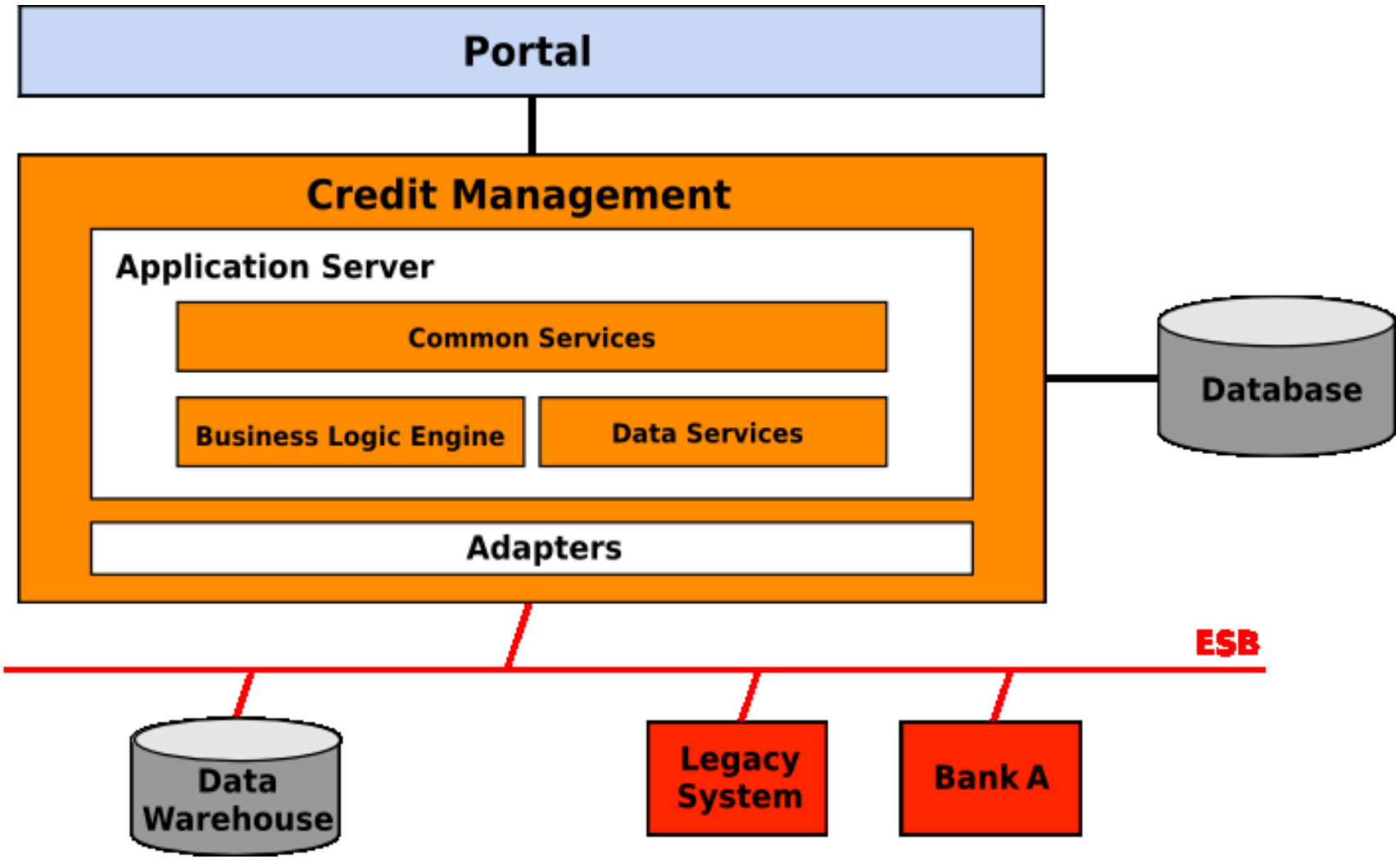
Signsoft GmbH

Java Forum Stuttgart 2006

Signsoft

Gliederung

- Apache Jakarta Commons
  - Motivation
  - Allgemeines
- Einsatz von Commons in Projekten
  - Utilities
  - Geschäftslogikmodul
  - Navigation im Objektgraphen


Portal
Credit Management
Application Server
Common Services
Business Logic Engine
Data Services
Adapters
Database
ESB
Data Warehouse
Legacy System
Bank A

Unterschiedliche Parametervalidierung

```
Credit createCredit(Customer customer, double amount) {
    ...
}

if (customer == null){
    throw new IllegalArgumentException("...");
}

Utlities.checkParameterNull(customer);

InputHelper.InputNotNull(customer);
```

Parametervalidierung mit Commons

```
Credit createCredit(Customer customer, double amount) {
    org.apache.commons.lang.Validate.notNull(customer, "customer
        musn't be null");
}
```

Über Jakarta Commons

- Projektstart Anfang 2001
- erstes Release commons-collections-1.0 im Juli 2001
- Open Source – Apache License 2.0
- Komponenten für die unterschiedlichsten Bereiche
- gutes und transparentes Qualitätsmanagement
- Support durch Mailinglist (commons-user@jakarta.apache.org)
- breiter Einsatz außerhalb der Jakarta-Projekte

Commons-Teile

- Commons Proper (33 Bibliotheken)
- Commons Dormant (16 Bibliotheken)
- Commons Sandbox (9 Bibliotheken)

Anwendungsgebiete I

- Ergänzungen zur Java-Plattform
  - Attribute
  - Codec
  - Collections
  - IO
  - Lang
  - Math
  - Pool
  - Primitives
  - Transaction
  - Validator
  - VFS

Anwendungsgebiete II

- Anwendungsinfrastruktur
  - CLI
  - Configuration
  - Daemon
  - DBCP
  - DbUtils
  - FileUpload
  - Launcher
  - Logging
  - Modeler

Anwendungsgebiete III

- XML
  - Digester
  - Jelly
  - (SCMXL)
- JavaBeans
  - BeanUtils mit BeanCollections
  - Betwixt
  - JXPath
- Pattern
  - Chain
  - Discovery

Anwendungsgebiete IV

- World Wide Web
  - Email
  - HttpClient
  - Net
- EL
- Jexl

Prämissen

- häufig gebrauchte Funktionalitäten
- Sprachspezifikation enthält nichts Passendes

Anforderungen

- Entwickler sollen keinen stupiden Quellcode schreiben
- zu pflegende Quellcodemenge minimieren
- stabiles Verhalten während des Lebenszyklusses der Anwendung

Lösungsansätze

Variante 1

eigene Utilities schreiben

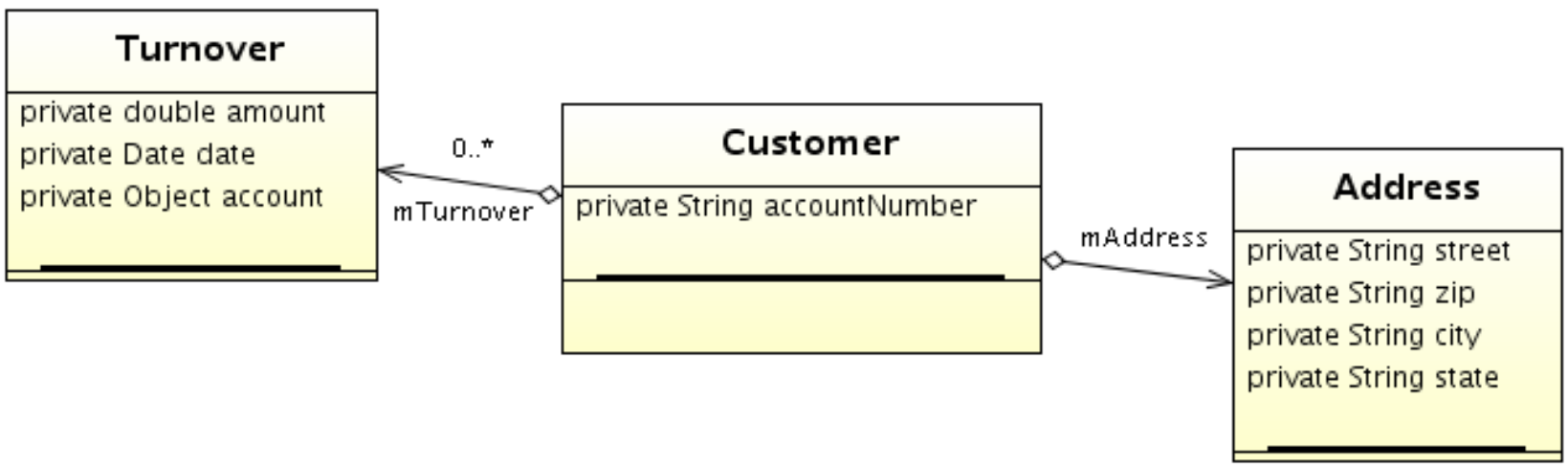
Variante 2

Templates, aus denen Klassen stets neu generiert werden

Variante 3

Jakarta Commons benutzen

Beispielobjekte



Anwendung von Commons Lang

equals() und toString()

```
public boolean equals(Object obj) {
    return EqualsBuilder.reflectionEquals(this,obj);
}

public String toString() {
    return ToStringBuilder.reflectionToString(this);
}
```

Anwendung von Commons Lang

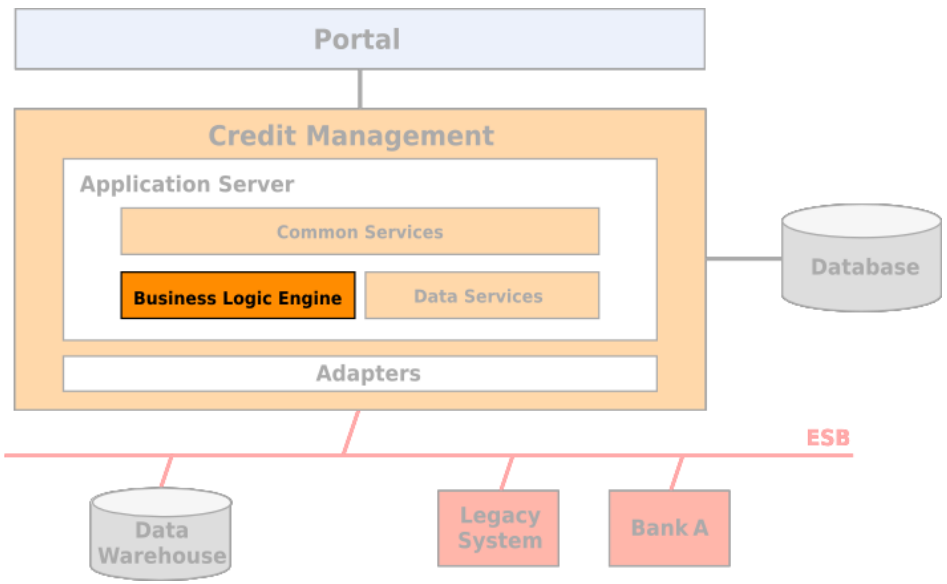
hashCode()

```
public int hashCode() {
    return new HashCodeBuilder(17, 37).
        append(street).
        append(zip).
        append(city).
        append(state).
        toHashCode();
    }
}
```

Grenzen

- laufzeitkritische Anwendungsteile
- Anwendungsarchitektur – Beispiel SecurityManager

Prämissen

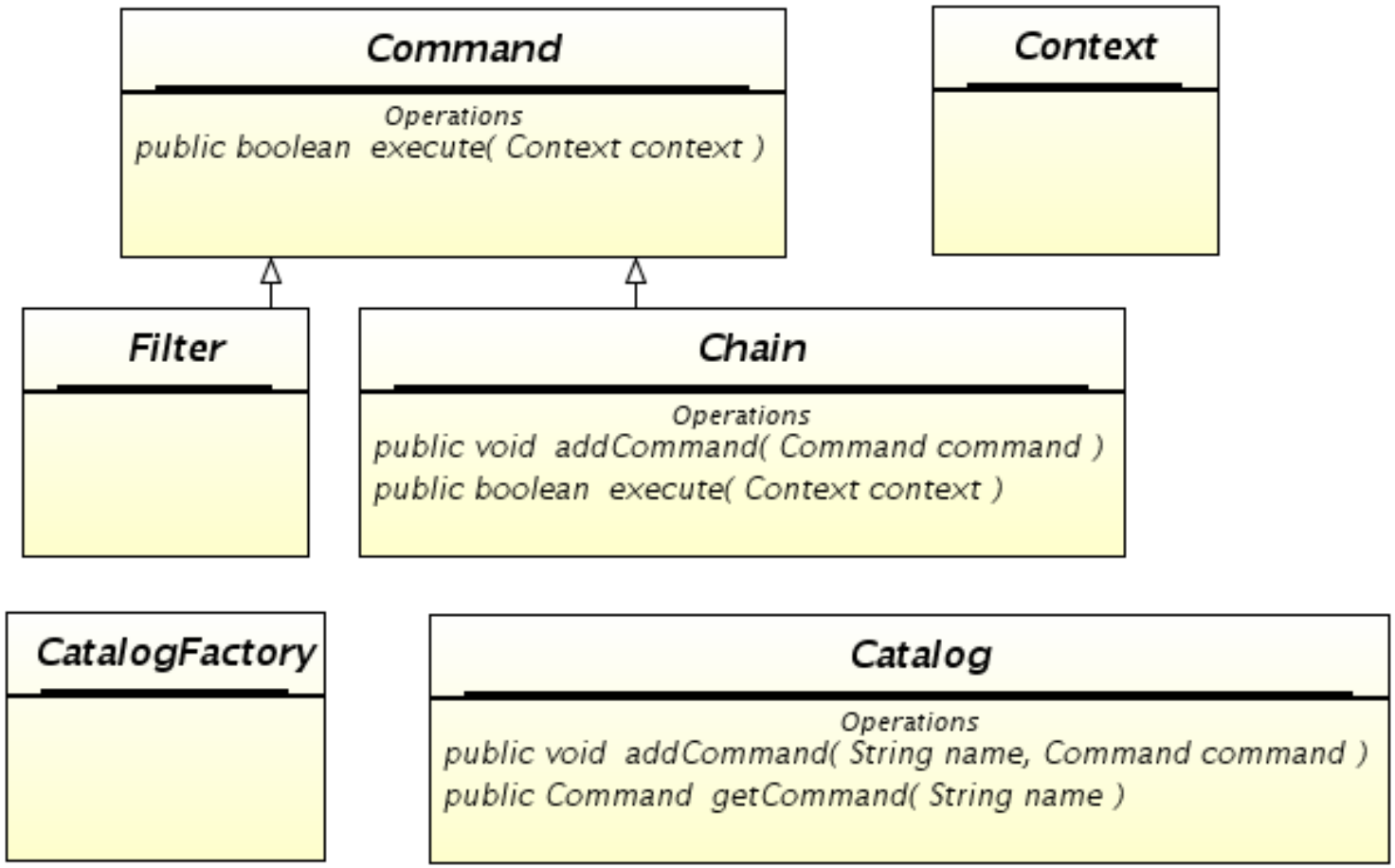


- kleine bis mittlere Menge an Geschäftsprozessen
- Prozesse haben eine gemeinsame Menge von Aktionen

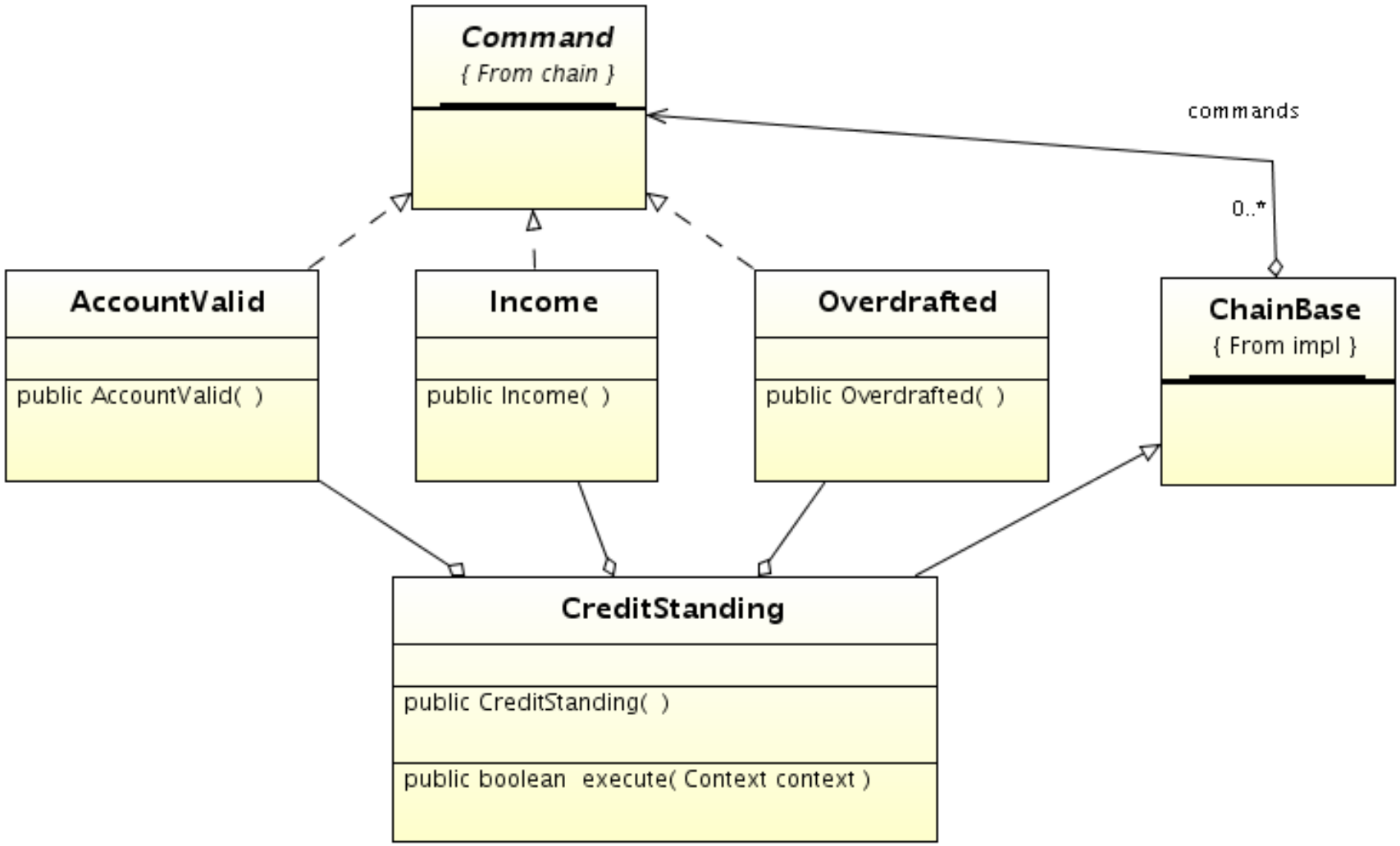
Anforderungen

- Erstellung von atomaren Geschäfslogikbausteinen
  - Überprüfungen
  - Berechnungen
  - Objekte verändern, erzeugen oder löschen
- Kombination zu komplexen Prozessen
- Prozesse müssen wiederverwendbar sein

Chain


Command
Operations
public boolean execute(Context context)
Context
Filter
Chain
Operations
public void addCommand(Command command)
public boolean execute(Context context)
CatalogFactory
Catalog
Operations
public void addCommand(String name, Command command)
public Command getCommand(String name)

Kreditwürdigkeit Kunde



Chain CreditStanding

```
class CreditStanding extends ChainBase {

    CreditStanding() {
        addCommand(new AccountValid());
        addCommand(new Income());
        addCommand(new Overdrafted());
    }

    public boolean execute(Context context) {
        String ok = context.get(Const.OK_KEY);
        ...
        return false;
    }
}
```

CreditBLCatalog

```
class CreditBLCatalog extends CatalogBase {

    private void registerCmds() {
        addCommand(Const.CREDIT_STANDING, new CreditStanding());
    }
}
```

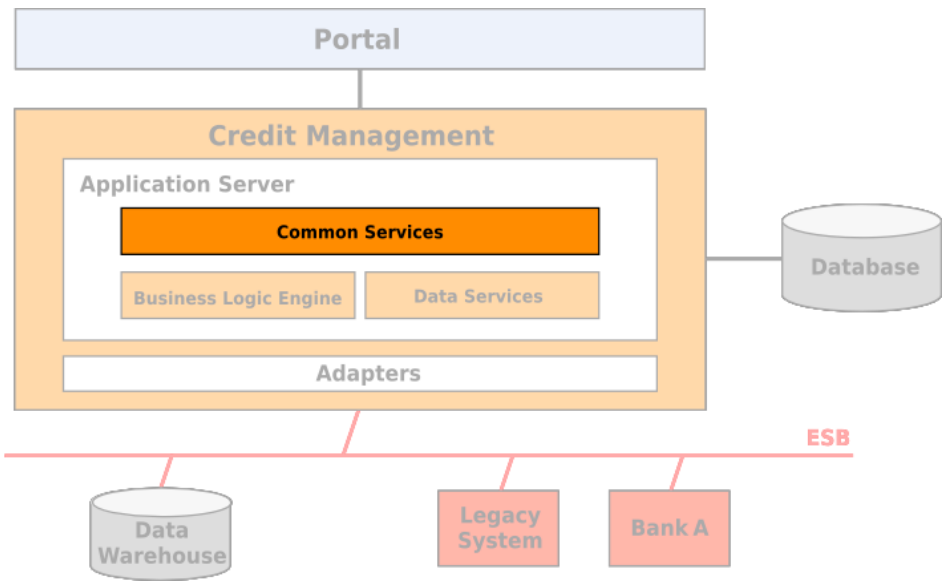
CreditEngine

```
class CreditEngine {
    ...
    CreditEngine() {
        mCatalog = new CreditBLCatalog();
    }
    ...
    public String creditStandingOK(Map map) {
        Context context = new ContextBase(map);
        boolean result = mCatalog.getCommand(Const.
            CREDIT_STANDING).execute(context);
        if (result) {
            return Const.CREDIT_STANDING_OK;
        }
        return Const.CREDIT_STANDING_BAD;
    }
}
```

Grenzen

- bei hochkonfigurierbarer Logik eine Rules Engine bevorzugen
- komplexe Prozesse mit vielen Daten können durch Context zu viel Speicher benötigen

Prämissen



- durch JDO-Benutzung ist ein großer Teil der benötigten Daten im Cache
- Geschäftsobjekte sind navigierbar
- Geschäftsobjekte sind kompatibel zur Bean-Spezifikation

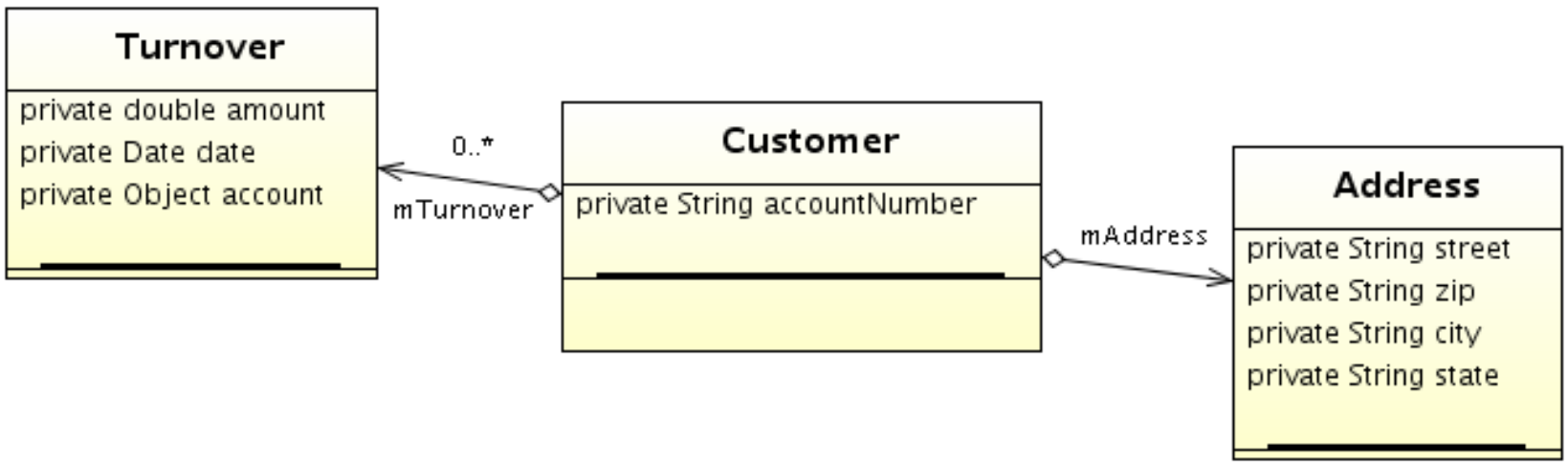
Anforderungen

- Zuordnung Oberflächenelement $\leftrightarrow$ Objektattribut
- Objektattribute laden und schreiben, gesteuert durch Anforderung der Oberfläche
- Dokumente mit Daten aus dem Objektgraphen erstellen
- Sortierung und Filterung von Objektlisten für Dokumente

Lösung

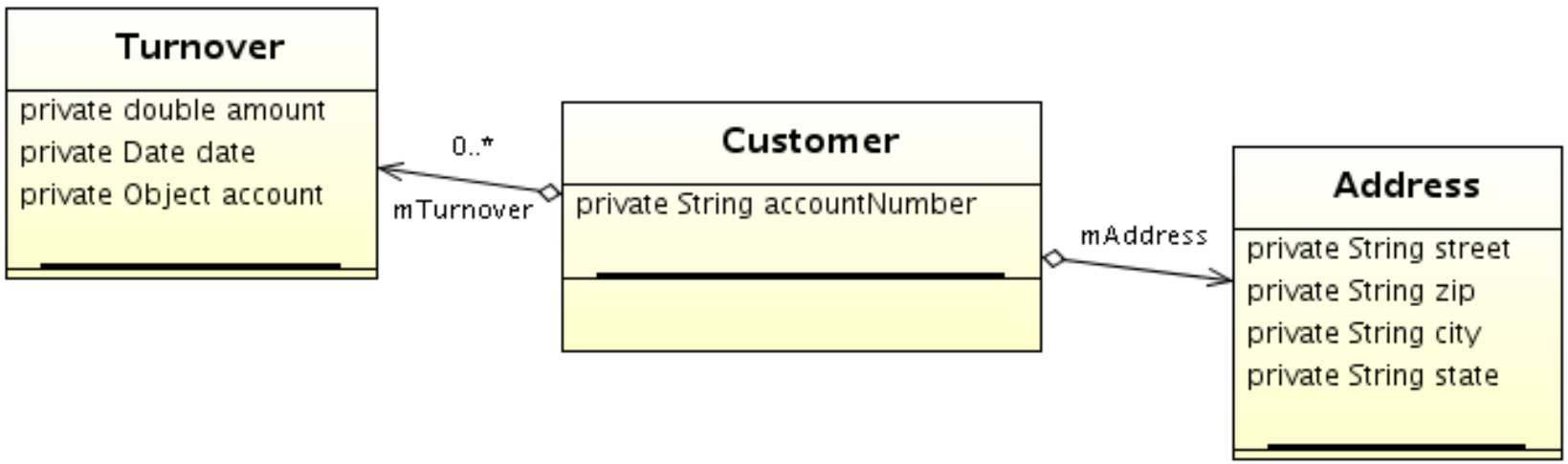
- JXPath
- Collections
  - ComparatorChain
  - FilterIterator
  - Predicates
- BeanUtils
  - BeanComparator
  - BeanPredicate

Reine Navigation




```
Customer customer = new Customer();
...
JXPathContext context = JXPathContext.newContext(customer);
String street = (String) context.getValue("address/street");
```

Navigation mit Listen




```
Date date = (Date) context.getValue("turnover[3]/date");
...
context.setValue("turnover[1]/account", new Account());
```

Sortierung

```
public void sort(Object[] array, OrderDescriptor[] order) {
    BitSet sortOrder = new BitSet(order.length);
    ArrayList toSort = new ArrayList();
    for (int i = 0; i < order.length; i++) {
        OrderDescriptor criteria = order[i];
        String beanKey = criteria.getKey().replace('/', '.');
        toSort.add(new BeanComparator(beanKey));
        if (!criteria.isAscending()) {
            sortOrder.flip(i);
        }
    }
    ComparatorChain multiSort = new ComparatorChain(toSort,
        sortOrder);
    Arrays.sort(array, multiSort);
}
```

Filterung – Predikate

```
public interface Predicate {
    boolean evaluate(Object obj);
}

public class Matches implements Predicate {
    Matches(Object value) {
        mExpression = (String) value;
    }
    public boolean evaluate(Object obj) {
        return ((String) obj).matches(mExpression);
    }
}

new BeanPredicate("address.city", new Matches("D.*"));
```

Filterung – FilterIterator

```
public Object[] filter(Object[] array, FilterDescriptor[] filter
    ) {
    ...
    Predicate predicate = buildPredicate(filter);
    ArrayList list = new ArrayList(Arrays.asList(array));
    FilterIterator filtered = new FilterIterator(list.iterator()
        , predicate);
    while (filtered.hasNext()) {
        filtered.next());
        ...
    }
    ...
}
```

Grenzen

- ungeeignetes Domänenmodell
- große Datenmengen

Zusammenfassung

Jakarta Commons bieten
vielfältige Bibliotheken mit unterschiedlichen Einsatzmöglichkeiten.

Jakarta Commons bieten nicht
eine fertige Lösung.

Signsoft

Weiterführende Literatur

- Timothy M. O'Brien
  Jakarta Commons Cookbook.
  O'Reilly, 2004
- `http://jakarta.apache.org/commons`
- `http://mail-archives.apache.org/mod_mbox/jakarta-commons-user`

Einsatz von Jakarta Commons

Jennifer Wagner

Signsoft GmbH

Java Forum Stuttgart 2006

Leipziger Str. 118
D-01127 Dresden

Tel.: +49 (0)351/89 45 30
Fax: +49 (0)351/89 45 329

www.signsoft.com
j.wagner@signsoft.com