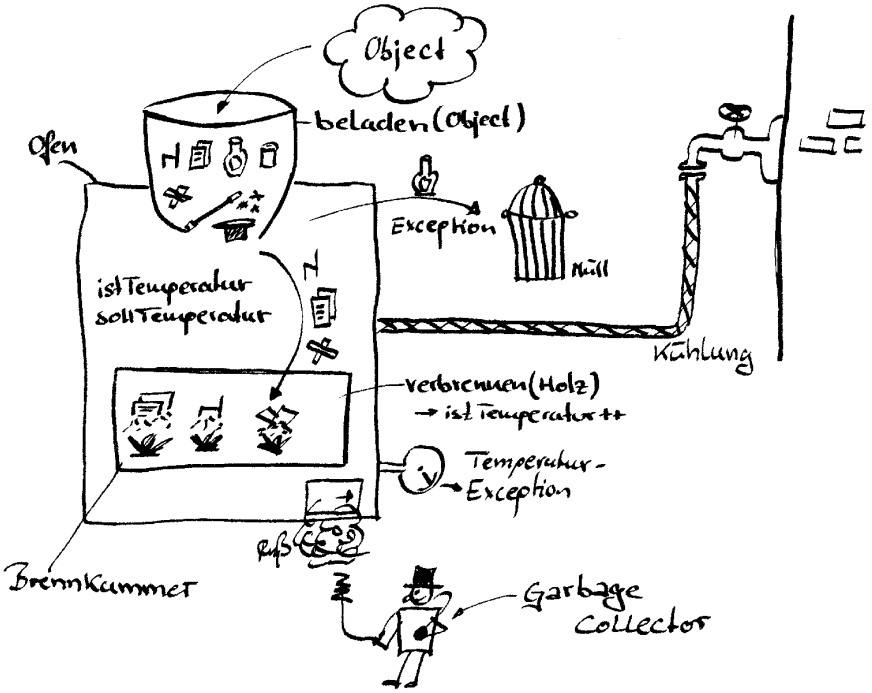
UEB

Übungen

1 Übungen

Als durchgängige Übung für das Seminar wird die Simulation eines Heizkraftwerks verwendet. Der Kraftwerksofen kann mit beliebigen Objekten beladen werden. Ein Filter sortiert nicht brennbare Objekte aus. Die brennbaren Objekte gelangen vom Ofen in die Brennkammer und verbrennen dort. Die Energie der verbrannten Brennelemente führt zu einem Anstieg der Ist-Temperatur. Falls die Ist-Temperatur größer als die eingestellte Soll-Temperatur ist, wird eine automatische unintelligente Kühlung die Ist-Temperatur um eine feste Kühl-Temperatur senken.

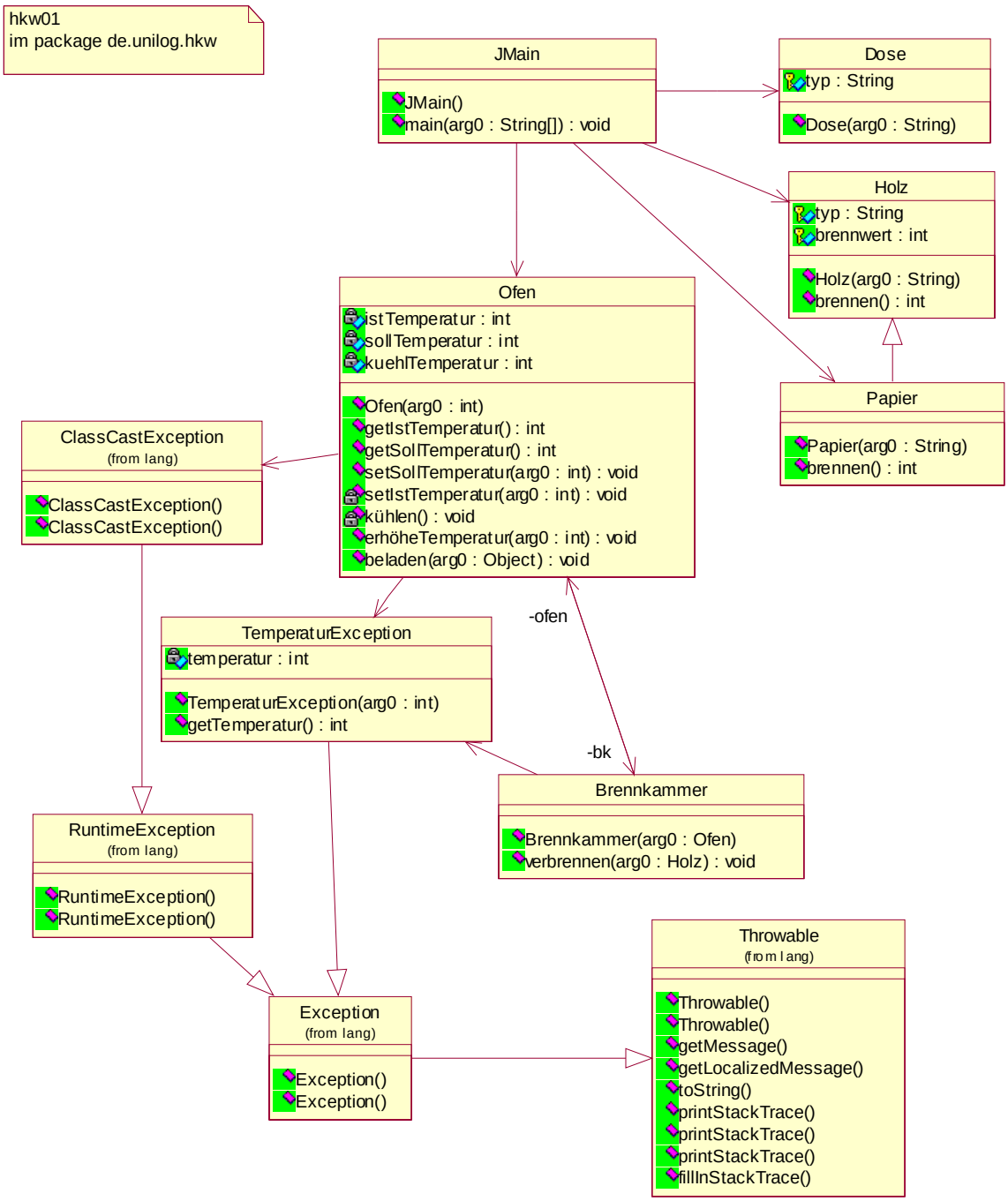
Die Simulation wird im Batch-Mode den Ofen und die Brennelemente erzeugen und die Brennelemente in den Ofen laden. Die Funktionsweise wird anhand von `System.out.println()` Ausgaben an der Konsole verfolgt werden.



1.1 Ofen, Brennkammer, Brennelemente und Exceptions

1. Erstellen Sie die Simulations-Startklasse **`JMain`** für die Heizkraftwerk-Applikation. Die Simulation soll durch entsprechende Ausgaben auf die Konsole (**`System.out.println()`**) lückenlos verfolgt werden können.
2. Erstellen Sie die Klassen für den Ofen und die Brennkammer.
3. Die Brennkammer wird zunächst als externe Klasse, später als innere Klasse entwickelt. Die Brennkammer wird zuerst nur zum Verfeuern von Holz ausgelegt.
4. Programmieren Sie die Brennelement-Klassen **`Holz`**.
5. Erzeugen Sie in **`JMain`** mehrere Brennelemente und einen Ofen. Beladen Sie den Ofen mit den Brennelementen.
6. Verbrennen Sie die Brennelemente. Dabei wird die Ofentemperatur entsprechend dem Brennwert erhöht.
   ➔ Musterlösung: **hkw00**
7. Programmieren Sie weitere Brennelement-Klassen **`Papier`** und **`Dose`**. Da **`Papier`** so nicht verbrennbar ist, wird die Klasse **`Papier`** von **`Holz`** abgeleitet. Warum kann **`Papier`** nur verbrannt werden?
8. Die Brennelemente sollen über einen „statischen Block“ verfügen, sodass beim ersten Zugriff auf **`Holz`** bzw. **`Papier`** eine entsprechende Meldung ausgegeben wird.
9. Fangen Sie die beim Beladen des Ofens auftretende **`ClassCastException`** ab und geben Sie eine entsprechende Meldung aus, wenn nicht brennbare Objekte (z.B. eine Dose) beladen werden.
10. Der Ofen kann zu heiß werden. Schreiben Sie eine eigene Exception-Klasse **`TemperaturException`**, die als Attribut die aktuelle Temperatur und eine Warnmeldung als weitere Information enthält.
11. Fangen Sie auch diese Exception nach dem Aufruf von **`verbrennen()`** ab und geben Sie die Meldung der Exception aus.
12. Schalten Sie bei Auftreten einer **`TemperaturException`** die automatische Kühlung des Systems an. Die Kühlung reduziert z.Z. nur die Ist-Temperatur um einen konstanten Wert (**`kühlTemperatur`**).

    ➔ Musterlösung: **hkw01**


hkw01
im package de.unilog.hkw
JMain
JMain()
main(arg0 : String[]) : void
Dose
typ : String
Dose(arg0 : String)
Holz
typ : String
brennwert : int
Holz(arg0 : String)
brennen() : int
Ofen
istTemperatur : int
sollTemperatur : int
kuehlTemperatur : int
Ofen(arg0 : int)
getIstTemperatur() : int
getSollTemperatur() : int
setSollTemperatur(arg0 : int) : void
setIstTemperatur(arg0 : int) : void
kühlen() : void
erhöheTemperatur(arg0 : int) : void
beladen(arg0 : Object) : void
Papier
Papier(arg0 : String)
brennen() : int
ClassCastException
(from lang)
ClassCastException()
ClassCastException()
TemperaturException
temperatur : int
TemperaturException(arg0 : int)
getTemperatur() : int
-ofen
-bk
Brennkammer
Brennkammer(arg0 : Ofen)
verbrennen(arg0 : Holz) : void
RuntimeException
(from lang)
RuntimeException()
RuntimeException()
Exception
(from lang)
Exception()
Exception()
Throwable
(from lang)
Throwable()
Throwable()
getMessage()
getLocalizedMessage()
toString()
printStackTrace()
printStackTrace()
printStackTrace()
fillInStackTrace()

1.2 Interface, Inner Classes, Garbage Collector und Reflection

1. Identifizieren Sie die unterschiedlichen Rollen bei der Entwicklung des Heizkraftwerkes und erstellen Sie unterschiedliche Namensräume (packages).
   `package ...hkw.brennelemente`; `//für Holz, ...`
   `package ...hkw.ofen`; `für Ofen, Brennkammer, ...`
   `JMain` bleibt als Testrahmen im `package de.integrata.hkw`.

2. Erstellen Sie ein neues `package ...hkw.moebel` mit den Klassen `Moebel`, `MetallSchrank` und `Sofa`. Das Sofa soll auch verbrennen.
   Machen Sie die Brennkammer flexibler, indem Sie die Brennelemente in der Brennkammer über das `Interface Brennbar` mit der Methode `brennen()` ansprechen. Erstellen Sie hierfür ein neues `package ...hkw.global`, das `Brennbar` enthält. Stellen Sie die Brennkammer-methode `verbrennen()` von `Holz` auf `Brennbar` um.

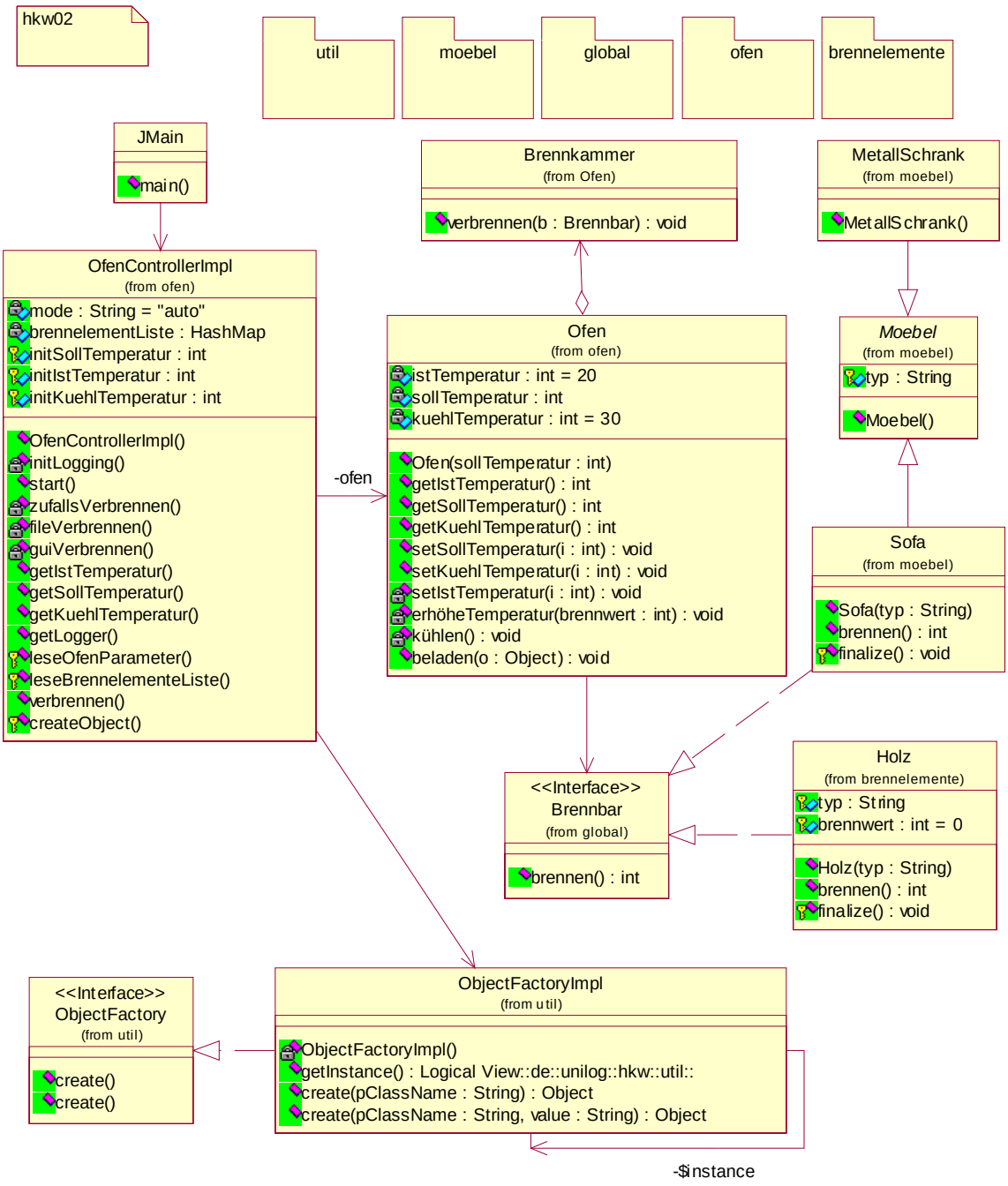
3. Zur Abbildung einer Komposition realisieren Sie nun die Brennkammer als „Innere Klasse“. Was hat dies für Auswirkungen auf die Klassen `Ofen` und `Brennkammer`.

4. Durch das Verbrennen in der `Brennkammer` verschwindet die Referenz auf das verbrannte Brennelement. Überschreiben Sie die `finalize()`-Methode und geben Sie dort eine geeignete Meldung beim Beseitigen durch den „Garbage Collector“ aus.

5. Um die Steuerung des Kraftwerks flexibler zu gestalten und von der Startklasse zu entkoppeln, wird eine Steuerungsklasse `OfenControllerImpl` geschaffen, die alle neu hinzukommenden Funktionen (Brennelemente in Collection-Klassen…) aufnimmt. `JMain` startet nur den OfenController.
   Benutzen Sie hierfür die Vorlage in
   `template/OfenControllerImpl.java`.

6. Als weiteres Feature soll die dynamische Erzeugung von Brennelementen über eine „Factory“ realisiert werden. Die generische `ObjectFactory` wird als `Singleton` implementiert und bietet die Methoden:
   `public Object create(String classname)` und
   `public Object create(String classname, String type)` an. Die `ObjectFactory` soll in das `package` ...`hkw/util` portiert werden und mit der Steuerungsklasse `OfenControllerImpl` integriert werden. Der Start der Simulation erfolgt weiterhin aus `JMain`.

   ➔ Musterlösung: **hkw02**


hkw02
util
moebel
global
ofen
brennelemente
JMain
main()
Brennkammer
(from Ofen)
verbrennen(b : Brennbar) : void
MetallSchrank
(from moebel)
MetallSchrank()
OfenControllerImpl
(from ofen)
mode : String = "auto"
brennelementListe : HashMap
initSollTemperatur : int
initIstTemperatur : int
initKuehlTemperatur : int
OfenControllerImpl()
initLogging()
start()
zufallsVerbrennen()
fileVerbrennen()
guiVerbrennen()
getIstTemperatur()
getSollTemperatur()
getKuehlTemperatur()
getLogger()
leseOfenParameter()
leseBrennelementeListe()
verbrennen()
createObject()
-ofen
Ofen
(from ofen)
istTemperatur : int = 20
sollTemperatur : int
kuehlTemperatur : int = 30
Ofen(sollTemperatur : int)
getIstTemperatur() : int
getSollTemperatur() : int
getKuehlTemperatur() : int
setSollTemperatur(i : int) : void
setKuehlTemperatur(i : int) : void
setIstTemperatur(i : int) : void
erhöheTemperatur(brennwert : int) : void
kühlen() : void
beladen(o : Object) : void
Moebel
(from moebel)
typ : String
Moebel()
Sofa
(from moebel)
Sofa(typ : String)
brennen() : int
finalize() : void
<<Interface>>
Brennbar
(from global)
brennen() : int
Holz
(from brennelemente)
typ : String
brennwert : int = 0
Holz(typ : String)
brennen() : int
finalize() : void
<<Interface>>
ObjectFactory
(from util)
create()
create()
ObjectFactoryImpl
(from util)
ObjectFactoryImpl()
getInstance() : Logical View::de::unilog::hkw::util::
create(pClassName : String) : Object
create(pClassName : String, value : String) : Object
-$instance

1.2.1 Dynamische Objekterzeugung durch die Reflection-API

Die Klasse Class

Mittels `Class.forName(String)` können dynamisch Klassen geladen werden. Durch Angabe des entsprechenden vollqualifizierten Klassennamen „`x.y.Z`“ wird die Klasse `Z.class` aus dem `package x.y` geladen. z.B.

```
Class.forName ("de.integrata.hkw.brennelemente.Holz");
```

Wenn die Klasse wie oben geladen ist, hat man über die Methoden von `Class` Zugriff auf alle Member (Felder und Methoden) dieser Klasse.

Es ist nun auch möglich, über die Methode `newInstance()` Objekte mit dem parameterlosen Konstruktor zu erzeugen. Falls zur Ereugung von Instanzen Konstruktoren mit Parametern benutzt werden sollen, kann die Methode `getConstructor()` verwendet werden, um den entsprechenden Konstruktor zu besorgen.

Zur Ermittlung der Konstruktoren werden die Typen der zu benutzenden Parameter und zur Instanzierung die entsprechenden Werte benötigt. Die Parametertypen der gewünschten Konstruktoren werden in einem `Class[]` angegeben. Für unsere Brennelemente-Klassen existieren jeweils Konstruktoren mit einem String-Parameter. Im Class-Array muss also der Typ der Klasse `java.lang.String` stehen. Dies kann folgendermaßen angegeben werden:

```
Class[] carr = {String.class};          //oder
Class[] carr = {Class.forName("java.lang.String")};

//für einfache Datentypen und Konstruktoren mit mehr Parametern z.B.:
Class[] carr = {int.class, char.class};
```

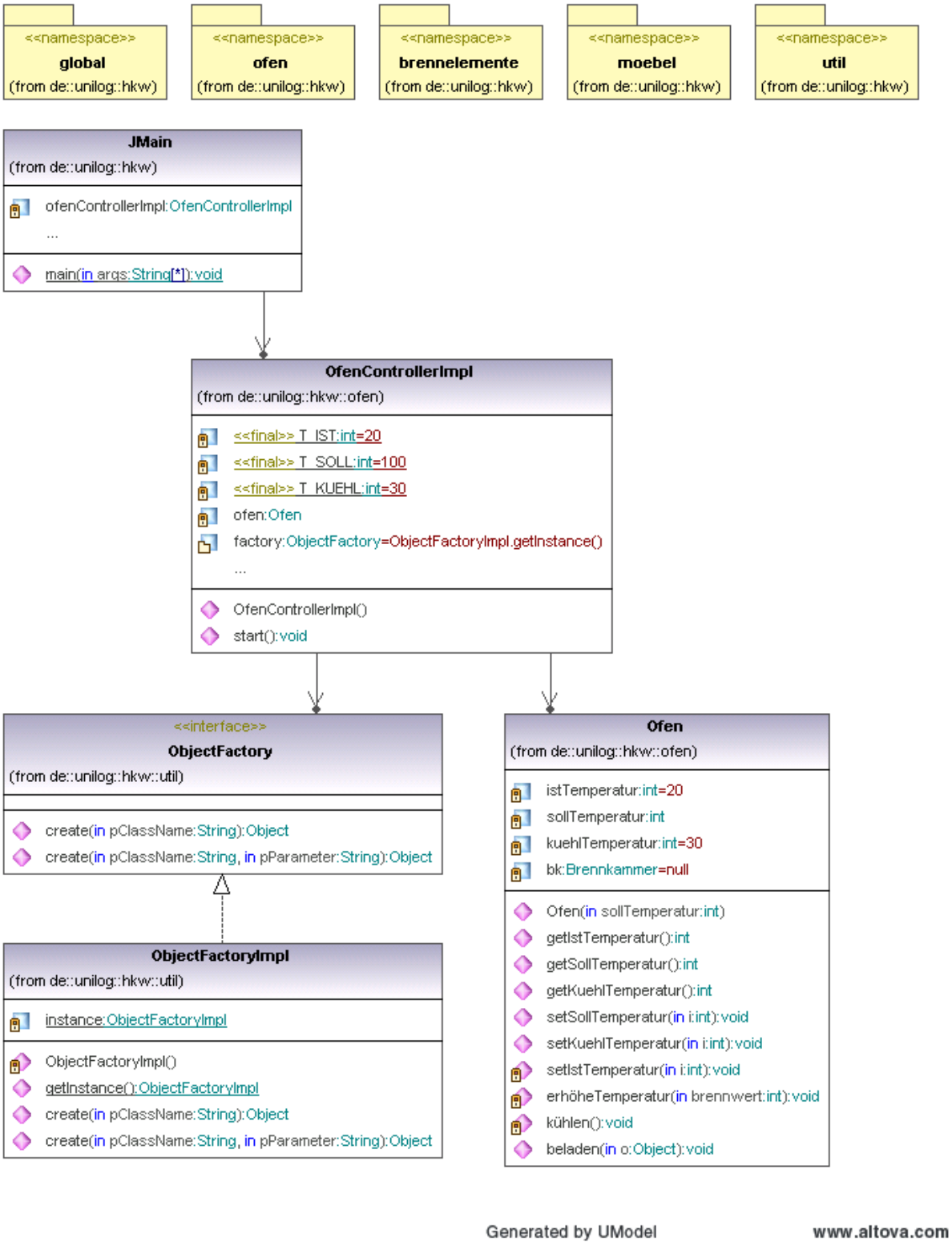
Beim Aufruf eines Konstruktors müssen diese formalen durch reale Parameterwerte ersetzt werden (also z.B. durch “Buche“, “Fichte“, ...). Die Parametertypen müssen in Typ und Anzahl mit den aktuellen Parameterwerten des verwendeten Konstruktors übereinstimmen.

```
Object[] oarr = {“Buche”};
//für einfache Datentypen und Konstruktoren mit mehr Parametern z.B.:
Object[] oarr = {new Integer(8), new Character(‘C’)};
```

Die Instanz wird dann durch

```
String cname = “de.integrata.hkw.brennelemente.Holz”;
Object o = Class.forName(cname).getConstructor(carr).newInstance(oarr);
```

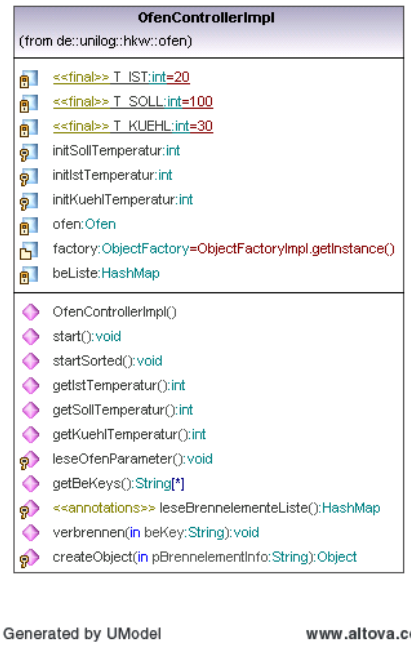
erzeugt.


<<namespace>>
global
(from de::unilog::hkw)
<<namespace>>
ofen
(from de::unilog::hkw)
<<namespace>>
brennelemente
(from de::unilog::hkw)
<<namespace>>
moebel
(from de::unilog::hkw)
<<namespace>>
util
(from de::unilog::hkw)
JMain
(from de::unilog::hkw)
ofenControllerImpl:OfenControllerImpl
...
main(in args:String[*]):void
OfenControllerImpl
(from de::unilog::hkw::ofen)
<<final>> T_IST:int=20
<<final>> T_SOLL:int=100
<<final>> T_KUEHL:int=30
ofen:Ofen
factory:ObjectFactory=ObjectFactoryImpl.getInstance()
...
OfenControllerImpl()
start():void
<<interface>>
ObjectFactory
(from de::unilog::hkw::util)
create(in pClassName:String):Object
create(in pClassName:String, in pParameter:String):Object
ObjectFactoryImpl
(from de::unilog::hkw::util)
instance:ObjectFactoryImpl
ObjectFactoryImpl()
getInstance():ObjectFactoryImpl
create(in pClassName:String):Object
create(in pClassName:String, in pParameter:String):Object
Ofen
(from de::unilog::hkw::ofen)
istTemperatur:int=20
sollTemperatur:int
kuehlTemperatur:int=30
bk:Brennkammer=null
Ofen(in sollTemperatur:int)
getIstTemperatur():int
getSollTemperatur():int
getKuehlTemperatur():int
setSollTemperatur(in i:int):void
setKuehlTemperatur(in i:int):void
setIstTemperatur(in i:int):void
erhöheTemperatur(in brennwert:int):void
kühlen():void
beladen(in o:Object):void
Generated by UModel
www.altova.com

1.3 Erweiterung der Controllerklasse und Einsatz von Collections

1. Die Controllerklasse **`OfenControllerImpl`** soll erweitert werden um die Funktionen zur:
   a. Konfiguration der Ofenparameter (Ist-, Soll- und Kühl-Temperatur),
   b. Verwaltung der Objekte in dynamischen Listen (Collection).

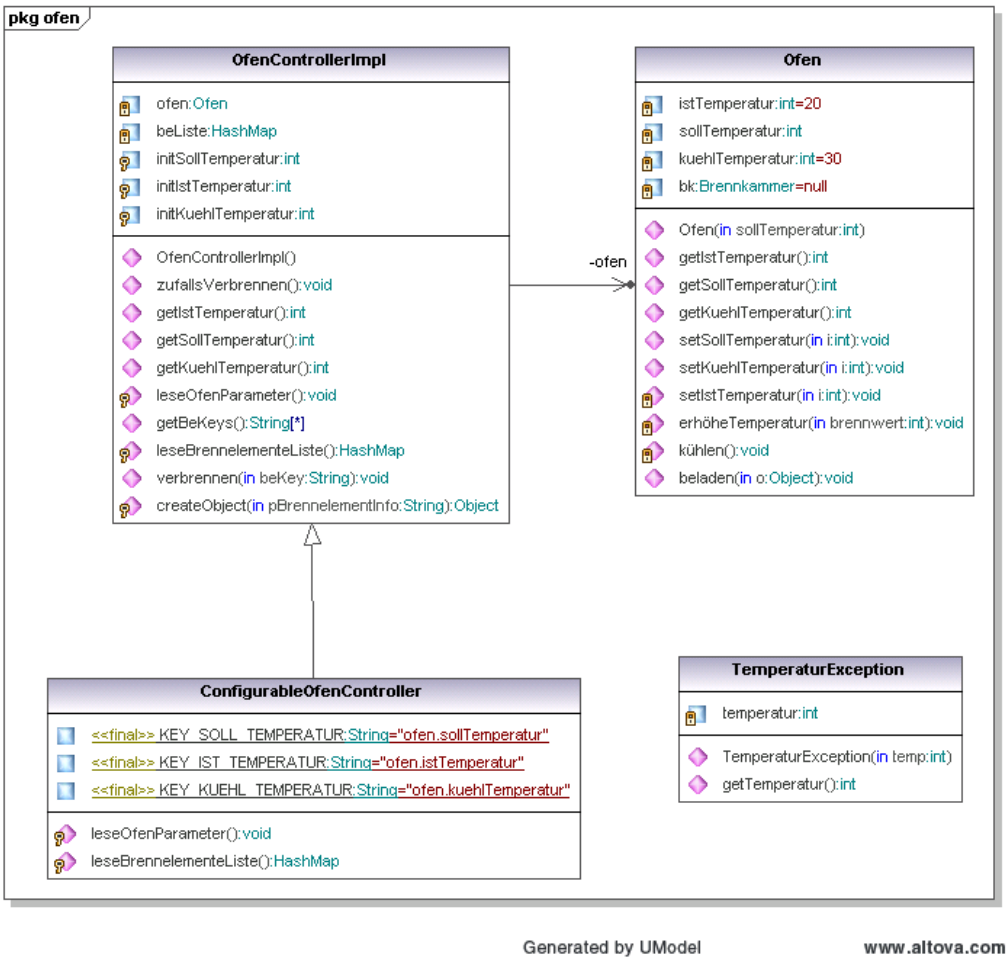
   Hierfür kann die Vorlage in
   `TEMPLATES/hkw03/ofen/OfenControllerImpl`
   analysiert und benutzt werden.
2. Realisieren/analysieren Sie eine Methode **`leseOfenparameter()`**, in der Default-Parameter aus einem initialisierten Array benutzt werden, um die Temperaturen des Ofens zu setzen.
3. Realisieren/analysieren Sie eine Methode **`leseBrennelementeliste()`** mit Default-Brennelementen. Die Brennelementliste soll einen logischen Namen und zugeordnet die Information zum dynamischen Erzeugen der Brennelemente mittels **`ObjecFactory`** (Klassenname und Konstruktorparameter) enthalten.
   Welche Collection ist dafür gut geeignet?
4. Sammeln Sie die erzeugten Brennelemente in einem Container und sortieren Sie die Elemente nach ihrem Brennwert.



➔ Musterlösung: **hkw03**

1.4 Ofen-Properties und Brennelemente-Liste aus Datei lesen

1. Lesen Sie die Ofenparameter aus einer Property-Datei.
2. Lesen Sie die benötigten Brennelemente-Informationen aus einer Datei ein. Die Datei soll Information über ein Brennelement jeweils in einer Zeile enthalten und besteht aus dem vollqualifizierten Klassennamen und dem Parameter für den Konstruktor durch ein Semicolon getrennt, z.B. für Holz:
   `de.integrata.hkw.brennelemente.Holz;Fichte`
3. Stellen Sie die interne Speicherung der Brennelemente von dem Array auf einen **`Vector/Hashtable`** bzw. **`ArrayList/Hashmap`** um.
4. Wie sieht die objektorientierte Realisierung der Erweiterungen aus? Ergeben sich Änderungen im Klassendiagramm?
5. Benutzen Sie die Vorlage in TEMPLATES/hkw04/ofen/ConfigurableOfenController



➔ Musterlösung: **hkw04**

1.5 Ofen mit statischer GUI

1. Erstellen Sie folgende statische GUI: Ein Fenster (Frame/Panel), das ein Label mit dem Kundennamen, ein Eingabe-Panel mit Bedienelementen und ein Label mit der aktuellen Uhrzeit und evtl. der Ofen-Ist-Temperatur enthält.

2. Das Eingabe-Panel hat ein `GridLayout` mit 2 Spalten und mindestens 5 Zeilen. Zuerst kommt ein `Label` mit Aufschrift "Identifikation", dann ein `TextField`, dann ein `Label` mit Aufschrift "Kennwort", noch ein `TextField`. Ein `Button` "kommen" und ein `Button` "gehen", eine `Choice` mit den Brennelementnamen und ein Button "beladen".

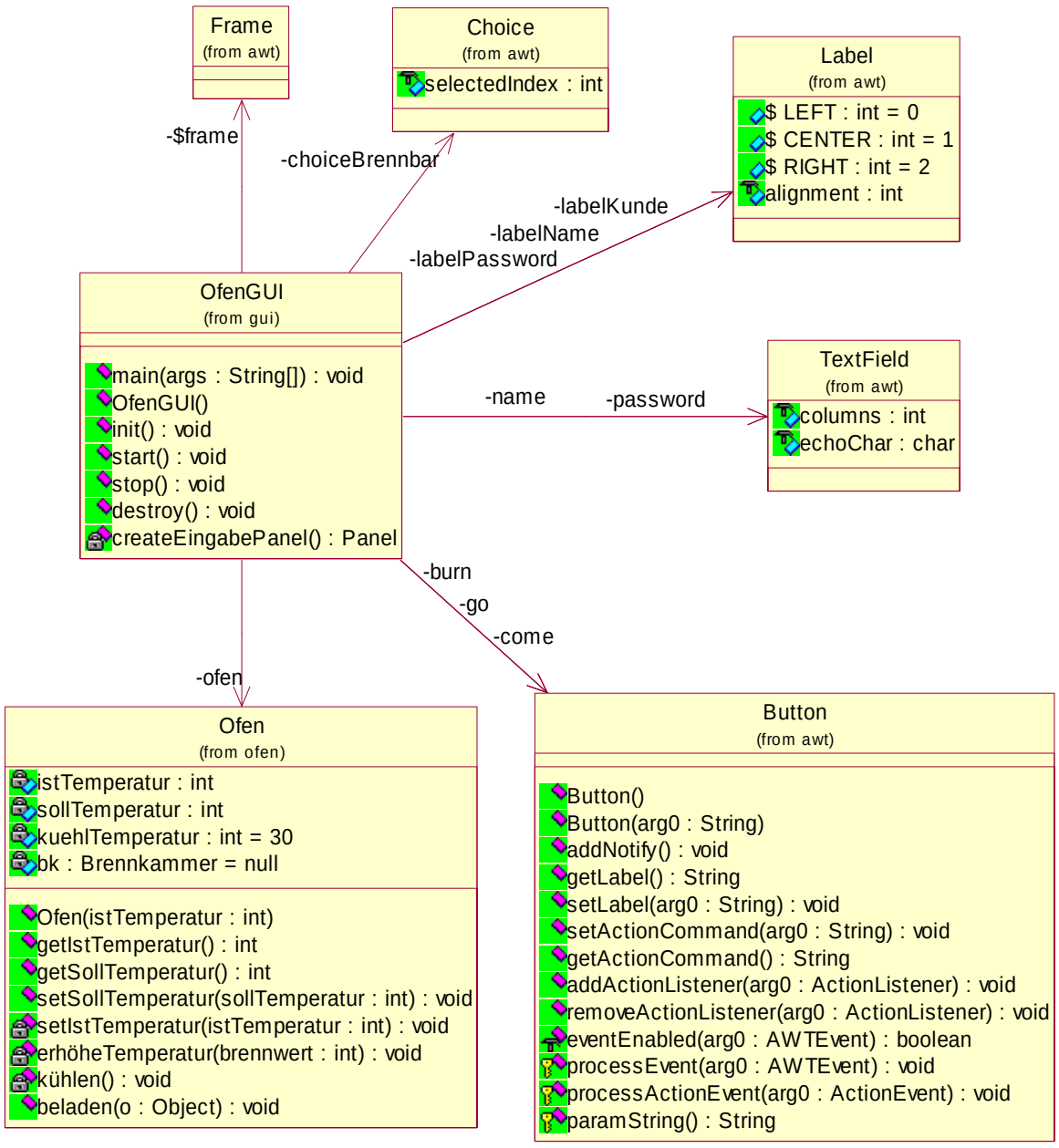
3. Das `EingabePanel` soll mit der Methode `createEingabePanel`() realisiert werden, um den Code des Eingabe-Panels wieder verwertbar (evtl. als eigene Klasse) in anderen Kraftwerken zu machen.

4. Wer fertig ist, überlegt sich, wie die Oberfläche optimiert werden kann, so dass sie ähnlich dem Bild aussieht.

5. (Ausrichtung, Farben, Fonts, Focus, Schaltflächen aktiviert/deaktiviert, `Gaps` im `LayoutManager`, Passwortfeld)

Hinweis: Es soll noch keinerlei Event-Behandlung durchgeführt werden, ausser dem Handler zum Schliessen des Fensters aus den Beispielen.

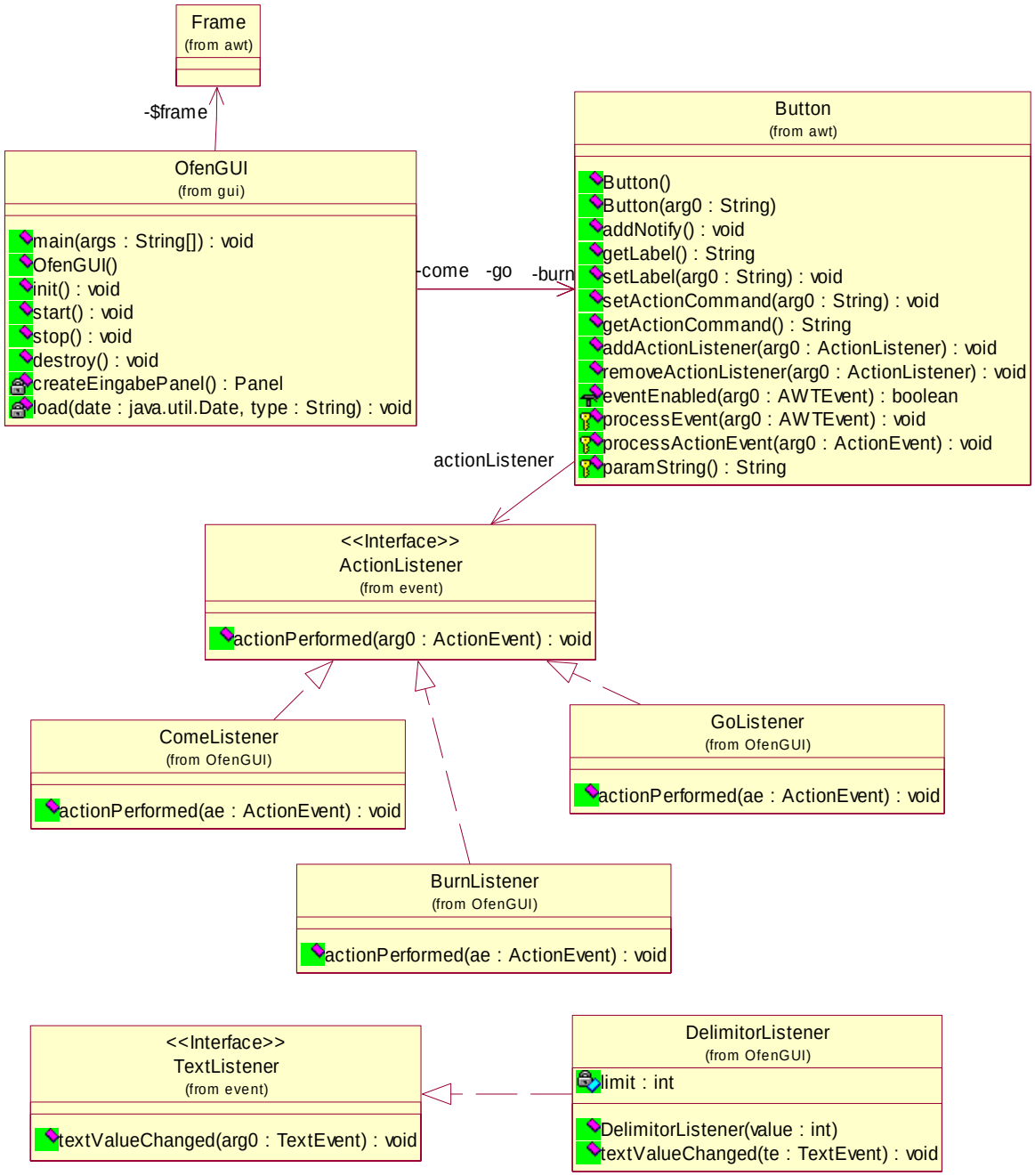
➔ Musterlösung: **hkw05**


Frame
(from awt)
Choice
(from awt)
selectedIndex : int
Label
(from awt)
$ LEFT : int = 0
$ CENTER : int = 1
$ RIGHT : int = 2
alignment : int
-$frame
-choiceBrennbar
-labelKunde
-labelName
-labelPassword
OfenGUI
(from gui)
main(args : String[]) : void
OfenGUI()
init() : void
start() : void
stop() : void
destroy() : void
createEingabePanel() : Panel
TextField
(from awt)
columns : int
echoChar : char
-name
-password
-burn
-go
-come
-ofen
Ofen
(from ofen)
istTemperatur : int
sollTemperatur : int
kuehlTemperatur : int = 30
bk : Brennkammer = null
Ofen(istTemperatur : int)
getIstTemperatur() : int
getSollTemperatur() : int
setSollTemperatur(sollTemperatur : int) : void
setIstTemperatur(istTemperatur : int) : void
erhöheTemperatur(brennwert : int) : void
kühlen() : void
beladen(o : Object) : void
Button
(from awt)
Button()
Button(arg0 : String)
addNotify() : void
getLabel() : String
setLabel(arg0 : String) : void
setActionCommand(arg0 : String) : void
getActionCommand() : String
addActionListener(arg0 : ActionListener) : void
removeActionListener(arg0 : ActionListener) : void
eventEnabled(arg0 : AWTEvent) : boolean
processEvent(arg0 : AWTEvent) : void
processActionEvent(arg0 : ActionEvent) : void
paramString() : String

1.6 Ofen mit dynamischer GUI

1. Erweitern Sie den Konstruktor von `OfenGUI` so, dass er einen anonymen `WindowAdapter` als `WindowListener` erzeugt und anmeldet. Dieser Listener soll auf das Event `windowClosing` reagieren und die Anwendung schließen.
2. Schalten Sie die Schaltflächen und die `Choice`-Box so an und aus, dass man den `Ofen` nur beladen kann, wenn man angemeldet ist.
3. Schreiben Sie drei "inner classes" als `ActionListener` für die Schaltflächen "kommen", "gehen" und "beladen". Diese beeinflußen den Zustand der übrigen Eingabekomponenten.
4. "beladen" soll den Mechanismus zum dynamischen Erzeugen der Objekte verwenden und dabei den selektierten Text in der `Choice`-Box benutzen um dynamisch ein Brennelement zu erzeugen.
5. Der `Ofen` soll mit den dynamisch erzeugten Objekten beladen werden.
6. Melden Sie die Listener bei den Schaltflächen an

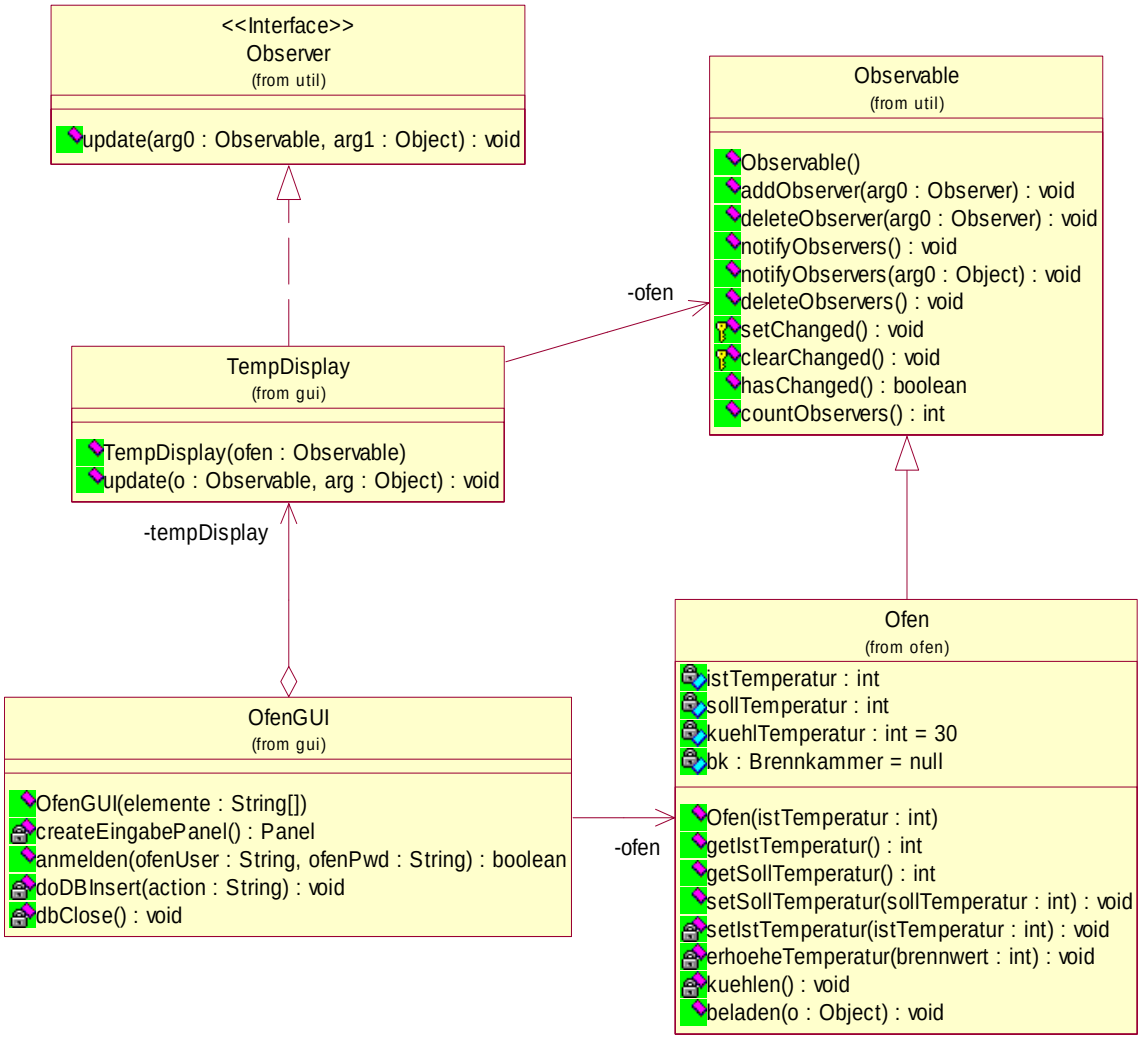
➔ Musterlösung: **hkw06**


Frame
(from awt)
-$frame
OfenGUI
(from gui)
main(args : String[]) : void
OfenGUI()
init() : void
start() : void
stop() : void
destroy() : void
createEingabePanel() : Panel
load(date : java.util.Date, type : String) : void
-come
-go
-burn
Button
(from awt)
Button()
Button(arg0 : String)
addNotify() : void
getLabel() : String
setLabel(arg0 : String) : void
setActionCommand(arg0 : String) : void
getActionCommand() : String
addActionListener(arg0 : ActionListener) : void
removeActionListener(arg0 : ActionListener) : void
eventEnabled(arg0 : AWTEvent) : boolean
processEvent(arg0 : AWTEvent) : void
processActionEvent(arg0 : ActionEvent) : void
paramString() : String
actionListener
<<Interface>>
ActionListener
(from event)
actionPerformed(arg0 : ActionEvent) : void
ComeListener
(from OfenGUI)
actionPerformed(ae : ActionEvent) : void
GoListener
(from OfenGUI)
actionPerformed(ae : ActionEvent) : void
BurnListener
(from OfenGUI)
actionPerformed(ae : ActionEvent) : void
<<Interface>>
TextListener
(from event)
textValueChanged(arg0 : TextEvent) : void
DelimitorListener
(from OfenGUI)
limit : int
DelimitorListener(value : int)
textValueChanged(te : TextEvent) : void

1.7 Ofen mit Model-View-Controler - Konzept

1. Die Java-Runtime-Bibliothek unterstützt das Observer Design-Pattern durch die Klasse `Observable` und das `Interface Observer`. Dies ermöglicht auf standardisierte Weise eine lose Kopplung zwischen Daten (dem Model) und mehreren voneinander unabhängigen Ansichten (Views).
2. Realisieren Sie durch eine neue Klasse `TempDisplay`, die von Frame abgeleitet wird, eine separate Temperaturanzeige
3. Die Anzeige soll Teil der `OfenGUI` sein und sich deshalb im `package gui` befinden. `TempDisplay` wird im Konstruktor von `OfenGUI` erzeugt.
4. `TempDisplay` soll ein `Observer` sein, der auf Änderungen der Ist-Temperatur des `Obserable-`Objektes `Ofen` reagiert und die aktuelle Temperatur anzeigt.

➔ Musterlösung: **hkw07**


<<Interface>>
Observer
(from util)
update(arg0 : Observable, arg1 : Object) : void
Observable
(from util)
Observable()
addObserver(arg0 : Observer) : void
deleteObserver(arg0 : Observer) : void
notifyObservers() : void
notifyObservers(arg0 : Object) : void
deleteObservers() : void
setChanged() : void
clearChanged() : void
hasChanged() : boolean
countObservers() : int
-ofen
TempDisplay
(from gui)
TempDisplay(ofen : Observable)
update(o : Observable, arg : Object) : void
-tempDisplay
OfenGUI
(from gui)
OfenGUI(elemente : String[])
createEingabePanel() : Panel
anmelden(ofenUser : String, ofenPwd : String) : boolean
doDBInsert(action : String) : void
dbClose() : void
-ofen
Ofen
(from ofen)
istTemperatur : int
sollTemperatur : int
kuehlTemperatur : int = 30
bk : Brennkammer = null
Ofen(istTemperatur : int)
getIstTemperatur() : int
getSollTemperatur() : int
setSollTemperatur(sollTemperatur : int) : void
setIstTemperatur(istTemperatur : int) : void
erhoeheTemperatur(brennwert : int) : void
kuehlen() : void
beladen(o : Object) : void

1.8 Ofen mit Thread für Anzeige/Uhr

1. Der Anwender, der die Kraftwerksanwendung benutzt, soll die aktuelle Uhrzeit (sekundengenau) immer eingeblendet bekommen. Erzeugen Sie hierzu durch Vererbung ein spezielles Label, das jede Sekunde die Uhrzeit neu ermittelt und mit dem Label anzeigt. **Tipp**: Die Klasse `DateLabel` ist von `java.awt.Label` abgeleitet und kann somit keine Sub-Klasse von `Thread` sein!
2. Benutzen Sie zur Formatierung der Uhrzeit und des Datums auf dem Label die Klasse `java.text.SimpleDateFormat`.
3. Sorgen Sie auch dafür, dass die Uhrzeit weder zu oft noch zu selten angezeigt wird. Hinweis: Die Klasse `java.lang.System` bietet eine effizientere Möglichkeit an, die Systemzeit zu ermitteln, als dies mit der Anweisung `new java.util.Date()` möglich ist. Welche?
4. Ein Gerüst des Labels, das Sie gegen Ihr vorhandenes austauschen können, ist bei den Beispielen zum Multithreading enthalten:

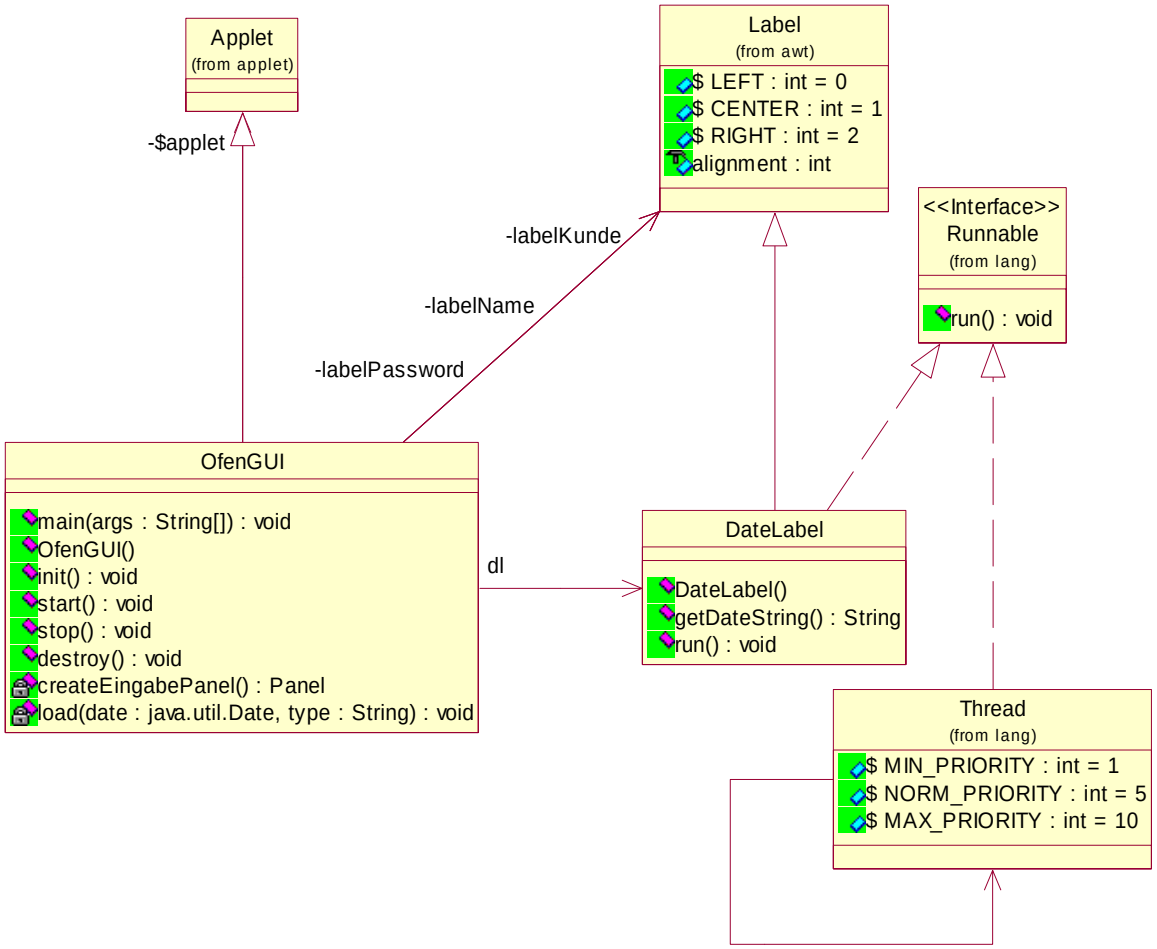
```
class DateLabel extends Label {

   private SimpleDateFormat sdf =
        new SimpleDateFormat("'Heute ist der' dd.MM.yyyy 'Zeit:' hh:mm:ss");
    private String dateString;
    private Date d = new Date(); // Uhrzeit ist statisch!

    public DateLabel() {
        ....
    } // DateLabel()

    /**
     * liefert das formatierte Datum in Form eines Strings
     */
    public String getDateString() {
        dateString = sdf.format(d);
        System.out.println(dateString);
        return dateString;
    }
}
```

➔ Musterlösung: **hkw08**


Applet
(from applet)
-$applet
Label
(from awt)
$ LEFT : int = 0
$ CENTER : int = 1
$ RIGHT : int = 2
alignment : int
<<Interface>>
Runnable
(from lang)
run() : void
-labelKunde
-labelName
-labelPassword
OfenGUI
main(args : String[]) : void
OfenGUI()
init() : void
start() : void
stop() : void
destroy() : void
createEingabePanel() : Panel
load(date : java.util.Date, type : String) : void
dl
DateLabel
DateLabel()
getDateString() : String
run() : void
Thread
(from lang)
$ MIN_PRIORITY : int = 1
$ NORM_PRIORITY : int = 5
$ MAX_PRIORITY : int = 10

1.9 Ofen mit Datenbankanbindung

1. Bisher kann jeder die Verbrennung steuern. Die Geschäftsführung verlangt, dass nur registrierte Benutzer die Anwendung nutzen dürfen.
2. Wir verwenden hierzu eine kleine Datenbank, **`Ofen.mdb`** für MS-Access bzw. HSQLDB oder MySQL. Die Datenbanken werden vom Trainer zur Verfügung gestellt bzw. auf dem lokalen PC eingerichtet. Die Datenbanken enthalten die Tabellen „**`Mitarbeiter`**“ mit Einträgen für Name und Kennwort und „**`Ofenprotokoll`**“ zur Protokollierung der Aktionen.
3. Falls MS-Access verwendet wird, muss über den Datenquellen-Administrator ein logischer Name (DSN = Data Set Name) **`ofen`** als ODBC-Datenquelle registriert werden.
4. Für den Zugriff werden die folgenden DB-Parameter verwendet:

```
#-------------------------------------------------------
# Parameterdatei fuer 3304 DB-Zugriff aus der Ofen-GUI
# 1.0.048 krg 30.04.08
#-------------------------------------------------------
#Parameter fuer MySQL
dbDriver=com.mysql.jdbc.Driver
dbURL=jdbc:mysql://localhost/ofen
dbUser=root
dbPwd=integrata
#Parameter fuer HSQLDB
#dbDriver=org.hsqldb.jdbcDriver
#dbURL=jdbc:hsqldb:hsql://localhost
#dbUser=sa
#dbPwd=
#Parameter fuer MS Access
#dbDriver=sun.jdbc.odbc.JdbcOdbcDriver
#dbURL=jdbc:odbc:ofen
```

6. Im EventHandler für den Kommen-Button muss die Datenbank abgefragt werden und die anderen Schaltflächen werden nur freigegeben, wenn der Benutzer sich mit richtigem Namen/Kennwort angemeldet hat.
7. Die Aktionen des Bedien-Personals sollen in der Datenbanktabelle „**`Ofenprotokoll`**“ protokolliert werden

Name	Aktion	Datum/Zeit
Marc	**Kommen**	**Fri Jan 14 15:23:01**
Marc	**Verbrenne brennelemente.Holz**	**Fri Jan 14 15:23:09**
Marc	**Verbrenne brennelemente.Dose**	**Fri Jan 14 15:23:19**
Marc	**Gehen**	**Fri Jan 14 15:23:22**

➔ Musterlösung: **hkw09**


<<Interface>>
Connection
(from sql)
createStatement() : Statement
prepareStatement(arg0 : String) : PreparedStatement
prepareCall(arg0 : String) : CallableStatement
nativeSQL(arg0 : String) : String
setAutoCommit(arg0 : boolean) : void
getAutoCommit() : boolean
commit() : void
rollback() : void
close() : void
isClosed() : boolean
getMetaData() : DatabaseMetaData
setReadOnly(arg0 : boolean) : void
isReadOnly() : boolean
setCatalog(arg0 : String) : void
getCatalog() : String
setTransactionIsolation(arg0 : int) : void
getTransactionIsolation() : int
getWarnings() : SQLWarning
clearWarnings() : void
<<Interface>>
Statement
(from sql)
executeQuery(arg0 : String) : ResultSet
executeUpdate(arg0 : String) : int
close() : void
getMaxFieldSize() : int
setMaxFieldSize(arg0 : int) : void
getMaxRows() : int
setMaxRows(arg0 : int) : void
setEscapeProcessing(arg0 : boolean) : void
getQueryTimeout() : int
setQueryTimeout(arg0 : int) : void
cancel() : void
getWarnings() : SQLWarning
clearWarnings() : void
setCursorName(arg0 : String) : void
execute(arg0 : String) : boolean
getResultSet() : ResultSet
getUpdateCount() : int
getMoreResults() : boolean
-dbConn
-dbStmt
OfenGUI
(from gui)
main(args : String[]) : void
OfenGUI()
init() : void
start() : void
stop() : void
destroy() : void
createEingabePanel() : Panel
load(date : java.util.Date, type : String) : void
dbConnect() : void
dbClose() : void
dbAktion(date : java.util.Date, action : String) : void
dbCheckUser(user : String, pwd : String) : boolean
<<Interface>>
ResultSet
(from sql)