

Octopus (<http://www.klasse.nl/english/research/octopus-intro.html>) ist ein Übersetzer von UML Klassenmodellen, die mit OCL versehen sind, nach Java Klassen, deren Methoden die in Modell spezifizierten Operationen, Vorbedingungen, Nachbedingung, Abfragen, und Invariante implementieren.

Solution:

```
<package> fuussball

<class> Spieler
<attributes>
+ name: String;
<endclass>

<class> Spiel
<attributes>
+ datum: Integer;
<endclass>

<class> Teilnahme
<endclass>

<associations>
+ Spieler.spieler [1..1]    <->  + Teilnahme.teilnahme [0..*]    ;
+ Spiel.spiel [1..1]      <->  + Teilnahme.teilnehmer [0..*]    ;
<endpackage>
```

```
package fuussball

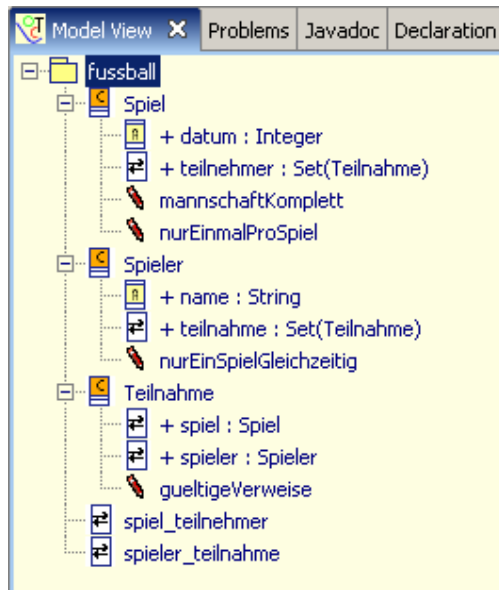
context Teilnahme
inv gueltigeVerweise :
(not spieler.ocllsUndefined()) and (not spiel.ocllsUndefined())

context Spieler
inv nurEinSpielGleichzeitig :
self.teilnahme.spiel->forall( s1, s2 |
(s1 <> s2) implies (
(s1.datum > s2.datum + 105*60 ) or
(s2.datum > s1.datum + 105*60 )
)
)

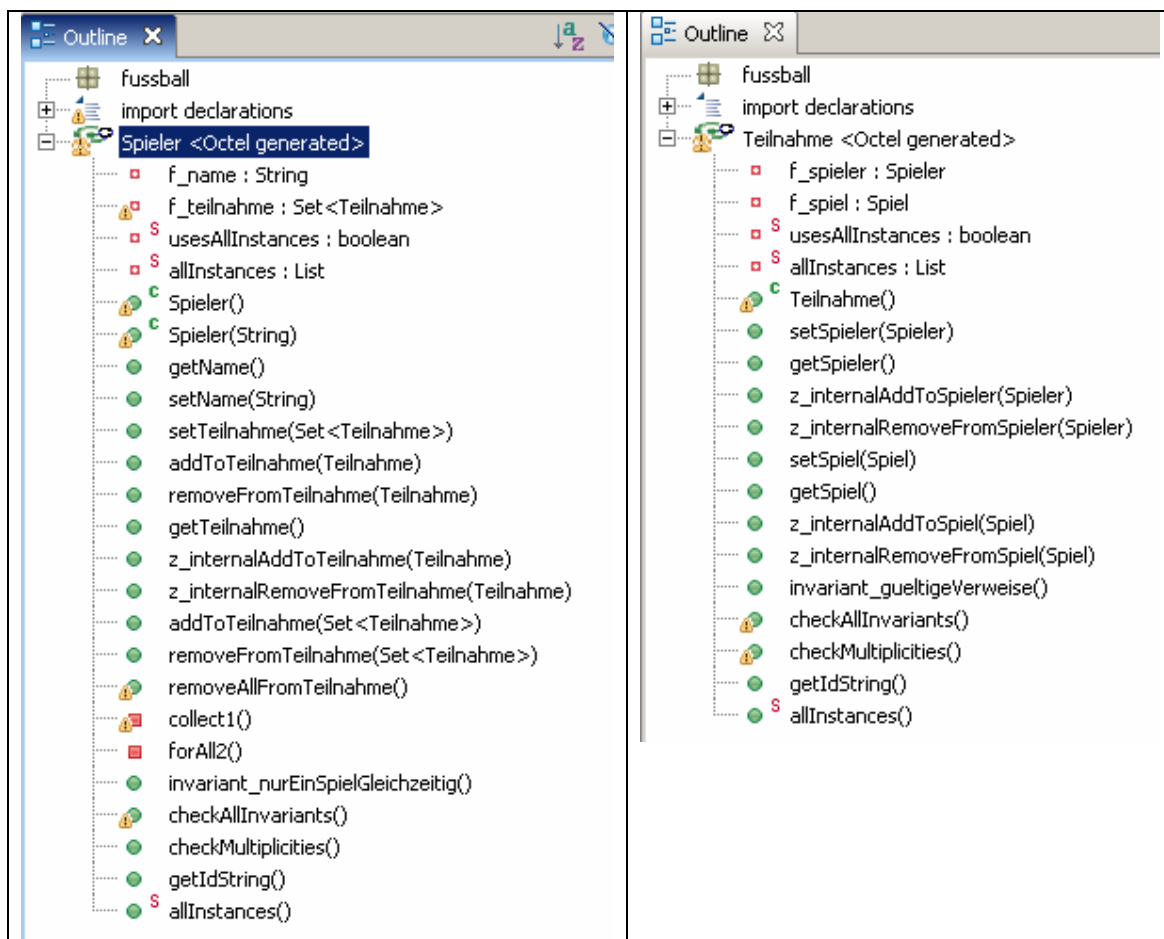
context Spiel
inv nurEinmalProSpiel :
self.teilnehmer = self.teilnehmer->asSet()

context Spiel
inv mannschaftKomplett :
self.teilnehmer->size() = 11
```

Das UML Klassenmodell besteht aus den Klassen:



Ein Überblick der generierten Java Klassen:



Invariante werden als Abfragen übersetzt, d.h. als side-effects-free Methode, z.B. gueltigeVerweise in Klasse Teilnahme:

```

/** Implements (not self.spieler.oclIsUndefined()) and not self.spiel.oclIs
Undefined()
*/
public void invariant_gueltigeVerweise() throws InvariantException {
    boolean result = false;
    try {
        result = (!(this.getSpieler() == null)) && (!(this.getSpiel() == null
));
    } catch (Exception e) {
        e.printStackTrace();
    }
    if ( ! result ) {
        String message = "invariant gueltigeVerweise ";
        message = message + "is broken in object ";
        message = message + this.getIdString();
        message = message + " of type " + this.getClass().getName() + "";
        throw new InvariantException(this, message);
    }
}

/** Checks all invariants of this object and returns a list of messages a
bout broken invariants
*/
public List checkAllInvariants() {
    List /* InvariantError */ result = new ArrayList /* InvariantError */();
;
    try {
        invariant_gueltigeVerweise();
    } catch (InvariantException e) {
        result.add(new InvariantError(e.getInstance(), e.getMessage()));
    }
    return result;
}

```

Die automatische Übersetzung von

```

context Spieler
inv nurEinSpielGleichzeitig :
self.teilnahme.spiel->forall( s1, s2 |
    (s1 <> s2) implies (
        (s1.datum > s2.datum + 105*60 ) or
        (s2.datum > s1.datum + 105*60 )
    )
)

```

```

/** Implements ->collect( i_Teilnahme : Teilnahme | i_Teilnahme.spiel )

// GENBY umlToJava.expgenerators.creators.LoopExpCreator::createCollect()
*/
private List<Spiel> collect1() {
    List /*(Spiel)*/ result = new ArrayList( /*Spiel*/);
    Iterator it = this.getTeilnahme().iterator();
    while ( it.hasNext() ) {
        Teilnahme i_Teilnahme = (Teilnahme) it.next();
        Object bodyExpResult = i_Teilnahme.getSpiel();
    }
}

```

```

        if ( bodyExpResult != null ) result.add( bodyExpResult );
    }
    return result;
}

/** Implements -
>forall( s1 : Spiel, s2 : Spiel | (s1 <> s2) implies (s1.datum > s2.datum + 105 * 60) or s2.da
tum > s1.datum + 105 * 60 )

// GENBY umlToJava.expgenerators.creators.LoopExpCreator::createForAll(), (iterVarNames.leng
th == 2)
*/
private boolean forAll2() {
    Iterator it = collect1().iterator();
    while ( it.hasNext() ) {
        Spiel s1 = (Spiel) it.next();
        Iterator it2 = collect1().iterator();
        while ( it2.hasNext() ) {
            Spiel s2 = (Spiel) it2.next();
            if ( (!(s1.equals(s2) ? (s1.getDatum() > (s2.getDatum() + (105 * 60))) || (s2.getDatum
() > (s1.getDatum() + (105 * 60)))) : true) ) {
                return false;
            }
        }
    }
    return true;
}

/** Implements self.teilnahme->collect( i_Teilnahme : Teilnahme | i_Teilnahme.spiel )-
>forall( s1 : Spiel, s2 : Spiel | (s1 <> s2) implies (s1.datum > s2.datum + 105 * 60) or s2.da
tum > s1.datum + 105 * 60 )
*/
public void invariant_nurEinSpielGleichzeitig() throws InvariantException {
    boolean result = false;
    try {
        result = forAll2();
    } catch (Exception e) {
        e.printStackTrace();
    }
    if ( ! result ) {
        String message = "invariant nurEinSpielGleichzeitig ";
        message = message + "is broken in object ";
        message = message + this.getIdString();
        message = message + " of type " + this.getClass().getName() + " ";
        throw new InvariantException(this, message);
    }
}
}

```

Weitere Links on Octopus:

- What associations in UML class models really mean
<http://www.devx.com/enterprise/Article/28528>
<http://www.devx.com/enterprise/Article/28576>
- The Royal & Loyal case study, with exercises:
<http://www.cse.dmu.ac.uk/~aoc/teaching-notes/Contents/CSCI3007/CSCI3007OCLtutorial.pdf>