

# Formal Object-oriented Software Development

---

Priv.-Doz. Dr. Rolf Hennicker

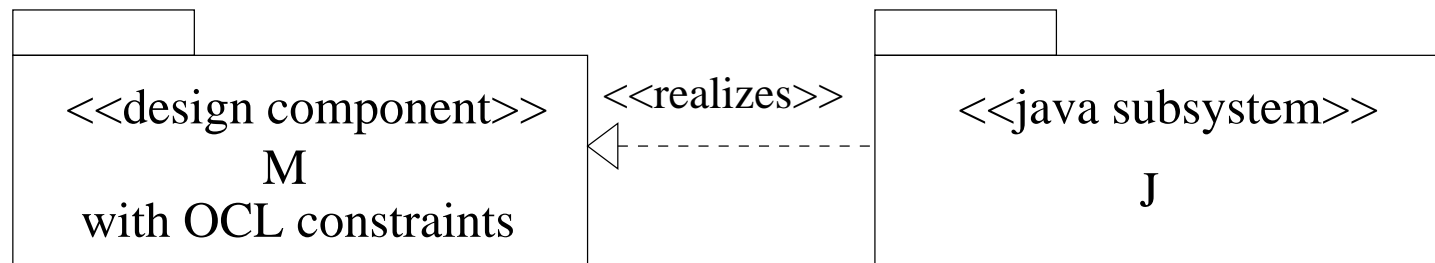
July 18, 2002

LMU

# Chapter 7: Realization of Class Specifications

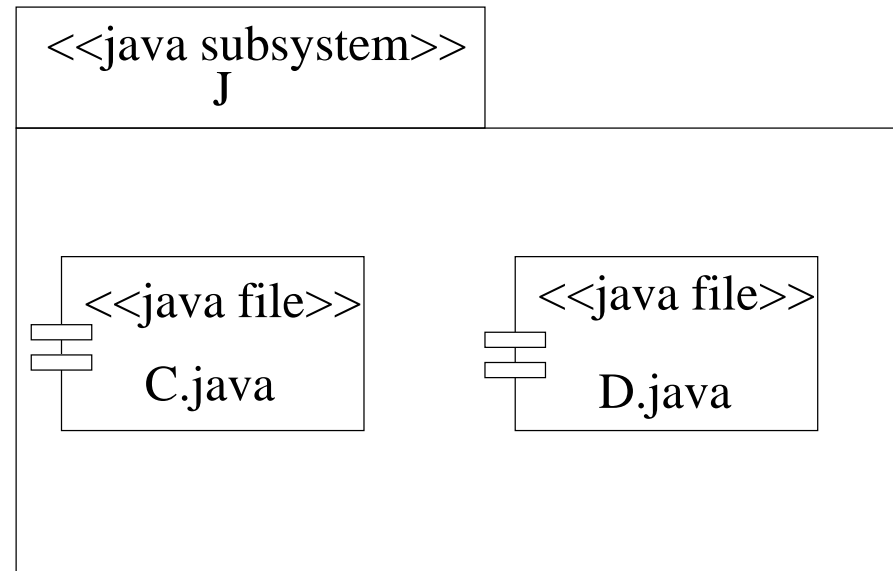
## Goals

- to know what a Java subsystem is
- to understand when a Java subsystem conforms to a class signature  $\Sigma_{\Delta}$
- to understand when a Java subsystem satisfies requirements for the Java method implementations and constructors
- to know the definition of a correct realization



- to understand the theorem on preservation of invariants

## 7.1 Java Subsystems



A Java subsystem consists of a set of files `C.java` such that each file `C.java` contains (w.l.o.g. exactly) one declaration of a Java class `C`.

**Definition 1 (Conformicity)**

Let  $\Sigma_{\Delta} = (S_{\Delta}, \leq, OP_{\Delta})$  be a class signature.

A Java subsystem  $J$  conforms to  $\Sigma_{\Delta}$  if

1. for each  $C \in Class_{\Delta}$  there exists a corresponding file  $C.java$  in  $J$ ,
2. for each  $(\_a : C \rightarrow T) \in A_{\Delta}$  there exists a corresponding instance variable (with the same name) in  $C.java$ ,
3. for each  $(op : C \times T_1 \times \dots \times T_n \rightarrow T) \in Q_{\Delta} \cup M_{\Delta}$  there exists a corresponding method in  $C.java$ , and  
for each constructor  $(create_C : T_1 \times \dots \times T_n \rightarrow C) \in Con_{\Delta}$  there exists a corresponding constructor  $C(T_1 x_1 \times \dots \times T_n x_n)$  in  $C.java$ ,
4. for each  $C, B \in Class_{\Delta}$ ,  $C \leq B$  and  $B$  a direct superclass of  $C$ , the Java class  $C$  (in  $C.java$ ) extends the Java class  $B$  (in  $B.java$ )
5. visibilities of attributes, roles and operations in  $M$  are preserved, i.e.
  - " $-$ " is mapped to "private",
  - " $\sim$ " is mapped to Java default visibility,
  - " $+$ " is mapped to "public" within a public Java class.

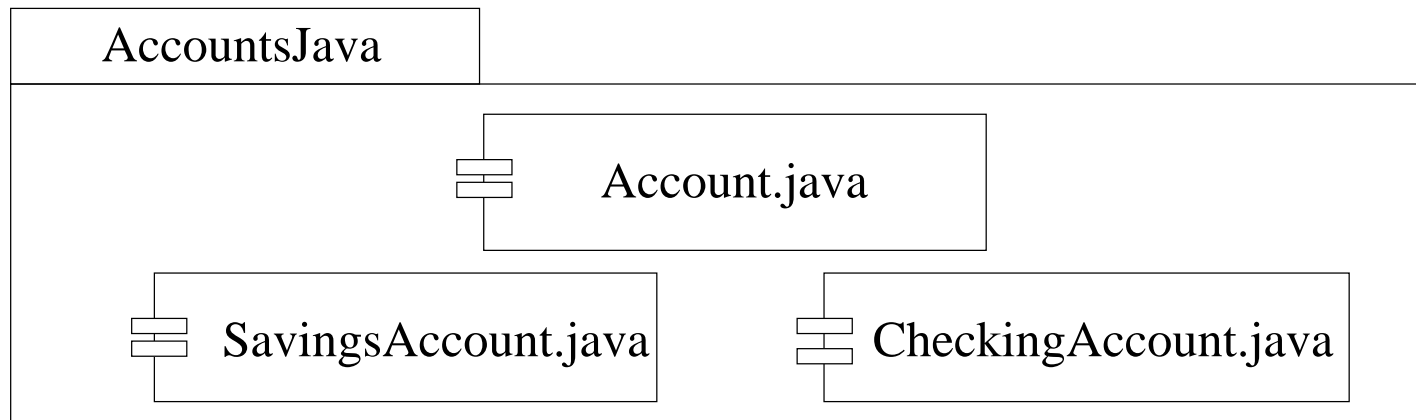
**Remark:**

1. Basic OCL types are renamed according to the syntax of Java,  
e.g.  $Integer \mapsto int$ .
2. The argument and result types of operations  $op \in Ops_{\Delta}$  are respected by their corresponding Java methods and constructors,  
e.g.  $(update : Account \times Real \rightarrow Void) \mapsto void\ update(double\ n)$ .
3. Collection types are replaced by Java container types (e.g. `Vector`).

**In particular:**

For each role name  $(\_ . a : C \rightarrow Set(T)) \in A_{\Delta}$  there exists a corresponding reference attribute (e.g. `Vector a;`) in `C.java`.

## Example 1 (AccountsJava)



```

public abstract class Account
{
    private double bal = 0;
    private double limit = 0;
    public void update(double n){
        bal = bal + n;
    }
    public double getBal(){
        return bal;
    }
}
  
```

```

public class SavingsAccount extends Account
{
    private double interestRate = 2.5;
    public void addInterest(){
        update(interestRate/100);
    }
}
  
```

```
public class CheckingAccount extends Account
{
    private double chargeRate = 8.5;
    private int countUpdates = 0;

    public void update(double n){
        super.update(n);
        countUpdates = countUpdates + 1;
    }

    public void payCharge(){
        super.update(-chargeRate);
    }
}
```

## 7.2 Satisfaction of Requirements by Java Subsystems

### **Notation:**

Let  $op$  be a non-abstract method of a Java class  $C$  or of a superclass of  $C$ . Then  $body_{C,op}$  denotes the body of  $op$  in  $C$  or in the least superclass of  $C$  which implements  $op$ .

### **Definition 2 (Satisfaction of requirements)**

Let  $\Sigma_{\Delta}$  be a class signature and  $J$  be a Java subsystem which conforms to  $\Sigma_{\Delta}$ . Assume that each variable  $self$  is renamed to  $this$ .

#### **1. Satisfaction of requirements for Java methods**

$$J \models \begin{array}{l} \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) : T \\ \text{pre : } Pre \\ \text{post : } Post \end{array}$$

*if for all  $\sigma^- \in Store_\Delta$  and  $\beta \in Valid(\sigma^-, Env_\Delta)$  the following holds:  
 if  $\llbracket Pre \rrbracket_{\beta, \sigma^-, \sigma^-, I} = true$  and  $body_{C, op}$  applied to  $\sigma^-$  terminates normally  
 with state  $\sigma$  and result value  $v$ ,  
 then  $\llbracket Post \rrbracket_{\beta[result \mapsto v], \sigma^-, \sigma, I} = true$  (and, if  $op$  is a query,  $\sigma = \sigma^-$ ).*

**Remark:**

- (a) *I is an arbitrary interpretation of queries satisfying the requirements.*
- (b) *The treatment of collection operations (like including) has still to be precised.*

## 2. Satisfaction of requirements for Java constructors

$$J \models \begin{array}{l} \text{context } C :: C(x_1 : T_1, \dots, x_n : T_n) \\ \text{pre : } Pre \\ \text{post : } Post \end{array}$$

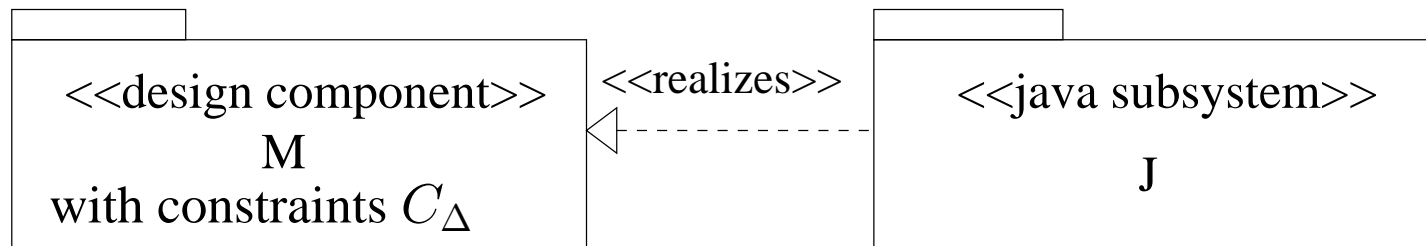
*if for all  $\sigma^- \in Store_\Delta$  and  $\beta \in Valid(\sigma^-, Env_\Delta)$  the following holds:  
 if  $\llbracket Pre \rrbracket_{\beta, \sigma^-, \sigma^-, I} = true$  and the statement  $C \ x = new \ C(x_1, \dots, x_n)$  applied to  
 $\sigma^-$  terminates normally with value  $v$  for the local program variable  $x$ ,  
 then  $\llbracket Post \rrbracket_{\beta[result \mapsto v], \sigma^-, \sigma, I} = true$ .*

## 7.3 Realization Relation

### Definition 3 (Realization relation)

Let  $ClassSpec = (\Delta, \Sigma_\Delta, C_\Delta)$  be a class specification  
(given by a design component  $M$  with constraints)  
and let  $(\Sigma_\Delta, REQ_\Delta)$  be the formal representation of  $ClassSpec$ .  
Let  $J$  be a Java subsystem.

*The realization relation*



*holds, if the following conditions are satisfied:*

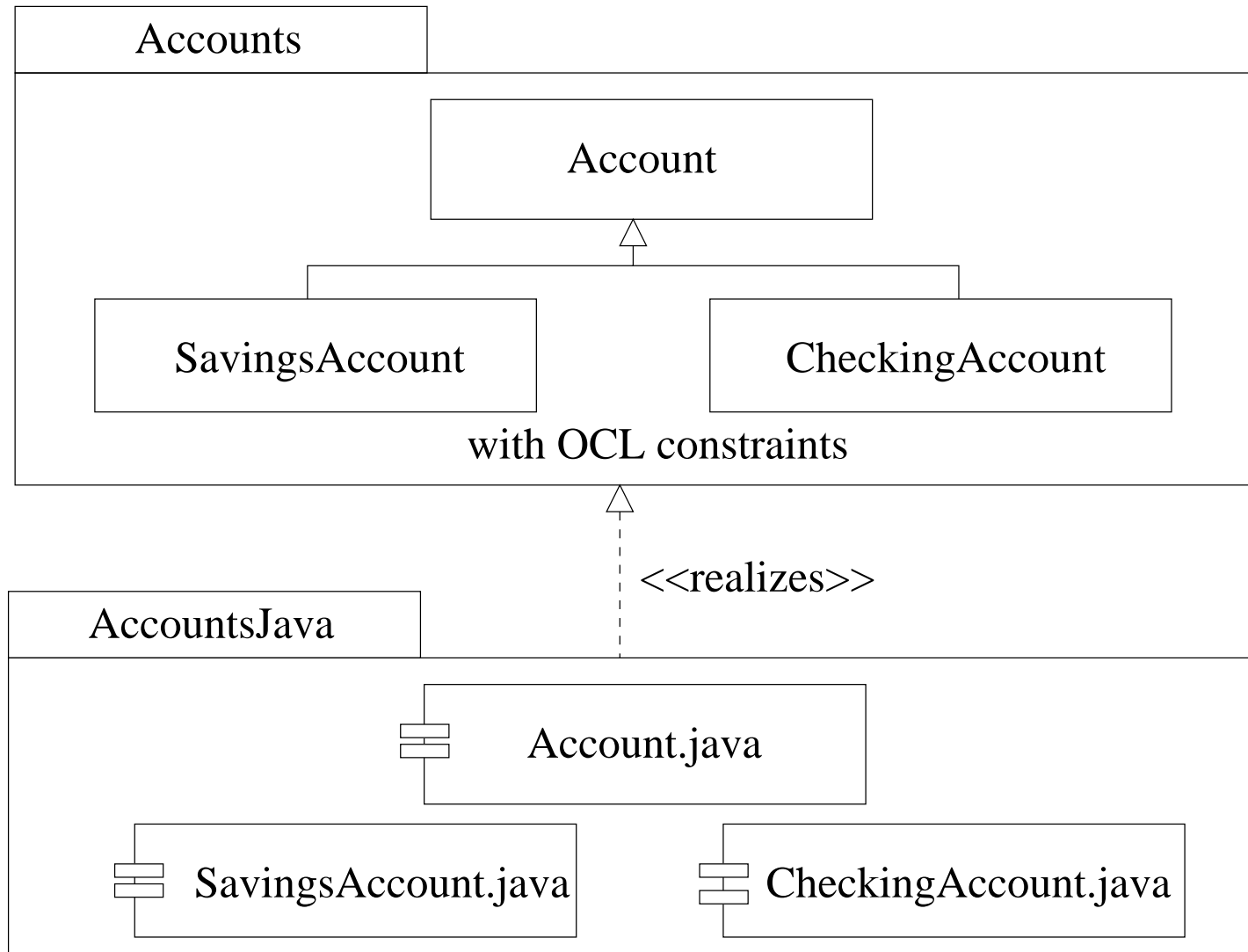
1. **Syntactic conditions**  $J$  conforms to  $\Sigma_{\Delta}$ .

2. **Semantic conditions**  $J \models REQ_{\Delta}$ , i.e.

$$J \models \begin{array}{l} \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) : T \\ \text{pre} : Pre \\ \text{post} : Post \end{array}$$

for each constraint in  $REQ_{\Delta}$ .

## Example 2 (AccountsJava realizes Accounts)



## 7.4 Preservation of invariants

### ***General assumptions***

- All attributes (i.e. instance variables) of a Java class are private
- Attribute access  $x.a$  is only done for  $x = this$ ,  
e.g. not allowed is

```
class C
{
    private int a;

    void op(C x)
    {
        x.a = 0;
    }
}
```

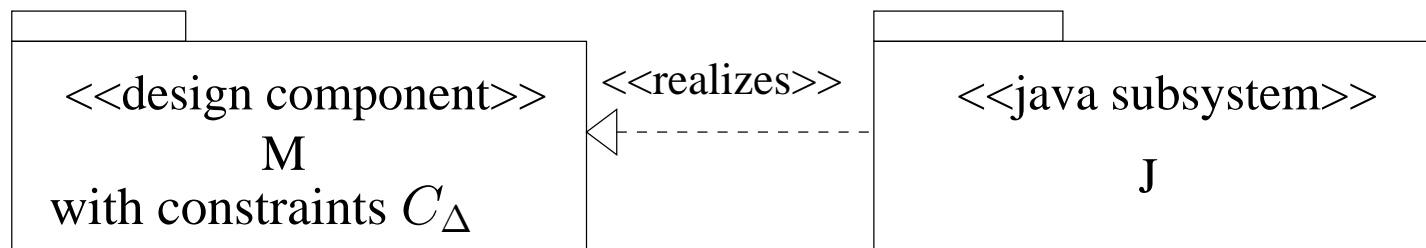
## Theorem 1 (Preservation of invariants)

Let  $ClassSpec = (\Delta, \Sigma_\Delta, C_\Delta)$  be a class specification (given by a design component  $M$ ) and let  $J$  be a Java subsystem.

### Particular assumptions

#### 1. Responsibility of the implementor:

(a)



*i.e. the realization is correct*

(b) Any non class private method or constructor occurring in  $J$  corresponds to an operation  $op \in Ops_\Delta$ .

## 2. Responsibility of the user:

*For any method call  $o.op(o_1, \dots, o_n)$  or constructor call  $new\ C(o_1, \dots, o_n)$  the precondition (given in  $C_\Delta$ ) of  $op(\dots)$  or of  $C(\dots)$  is satisfied in the actual state  $\sigma^-$ . Moreover, if  $o_i = this$  for some  $i \in \{1, \dots, n\}$  and  $this$  is of type  $C$ , then  $INV_C$  is satisfied for  $o_i$  in  $\sigma^-$ .*

*Then the following holds:*

- 1. Any call  $obj.opn(obj_1, \dots, obj_k)$  or  $new\ C(obj_1, \dots, obj_k)$  of a component private operation or constructor preserves all class invariants.*
- 2. Any call  $obj.opn(obj_1, \dots, obj_k)$  or  $new\ C(obj_1, \dots, obj_k)$  of a component public operation or constructor preserves all class invariants and the component invariant.*

**Proof:**

W.l.o.g. let  $opn$  be a component private operation. Let  $\sigma^- \in Store_\Delta$  such that the class invariants are satisfied for  $obj$  and for  $obj_1, \dots, obj_k$ . By assumption 2, the precondition of  $opn$  is satisfied and, by assumption 1,  $J \models PREINV_{C::op}$ .

Hence, if  $body_{C,op}$  applied to  $\sigma^-$  terminates with state  $\sigma$ , then the class invariant  $INV_C$  is satisfied for  $obj$  in  $\sigma$ .

It remains to prove that also all class invariants for objects  $o \neq obj$  are preserved. For this it is sufficient to prove the following Lemma:

**Lemma 1 (Preservation of invariants)**

*Let the assumptions 1 and 2 of the theorem be satisfied.*

*For each  $C \in Class_\Delta$ ,  $\sigma^- \in Store_\Delta$ ,  $obj \in_{\sigma^-} C$  such that*

*$\llbracket INV_D \rrbracket_{\beta[this \mapsto o], \sigma^-, \sigma^-, I} = true$  for all  $D \in Class_\Delta$  and*

*for at least all objects  $o \neq obj$ ,  $o \in_{\sigma^-} D$ , the following holds:*

*If a call  $obj.opn(obj_1, \dots, obj_n)$  or  $new C(obj_1, \dots, obj_n)$  terminates normally with state  $\sigma^-$ , then*

*$\llbracket INV_D \rrbracket_{\beta[this \mapsto o], \sigma, \sigma, I} = true$  for all  $D \in Class_\Delta$  and*

*for at least all objects  $o \neq obj$ ,  $o \in_\sigma D$ .*

**Proof of the Lemma:**

By induction on the depth  $n$  of nested method or constructor calls.

**Case 0:**  $n = 0$ .

According to the general assumptions, only the state of  $obj$  can have changed. Hence all invariants for all objects  $o \neq obj$  remain valid.

**Case 1:**  $n > 0$ .

Let  $obj'.opn'(obj'_1, \dots, obj'_{k'})$  or  $new C'(obj'_1, \dots, obj'_{k'})$  be a nested call performed during the execution of the body of  $opn$  or  $new C(\dots)$ . W.l.o.g. assume that no further call is performed before and afterwards on the same level  $n = 1$  of nesting. According to the general assumptions all objects  $o \neq obj$  have not changed their state, i.e. satisfy their invariant before the nested call.

**Case 1.1:**  $obj' = obj$ .

Then, by induction hypothesis, all objects  $o \neq obj' = obj$  satisfy their invariants after execution of the nested method call and therefore, by the general assumptions, also after execution of the overall call.

**Case 1.2:**  $obj' \neq obj$ .

Then  $obj'$  satisfies its invariant. Moreover, all actual parameters  $obj'_1, \dots, obj'_{k'}$  satisfy their invariants (even, according to assumption 2, if one object  $obj'_i = obj$ ). Since, by assumption,  $J \models PREINV_{C'::opn'}$  the object  $obj'$  satisfies its invariant after execution of the nested call. Moreover, by induction hypothesis, also all other objects  $o \neq obj$  satisfy their invariants after the nested call and therefore, by the general assumption, also after the execution of the overall call.

□

## 7.5 Summary

- A Java subsystem consists of a set of Java files containing declarations of Java classes.
- If a Java subsystem  $J$  conforms to a class signature  $\Sigma_{\Delta}$  then
  - OCL-expressions can be evaluated according to a given system state during the execution of a Java program and
  - the satisfaction of constraints for Java method implementations and constructors can be defined.
- A Java subsystem  $J$  realizes a class specification  $ClassSpec = (\Delta, \Sigma_{\Delta}, C_{\Delta})$  if
  - $J$  conforms to  $\Sigma_{\Delta}$
  - $J \models REQ_{\Delta}$  where  $(\Sigma_{\Delta}, REQ_{\Delta})$  is the formal representation of  $ClassSpec$ .
- If implementors and users respect their responsibilities then at any time before and after execution of a component private (component public) operation the system is in a consistent state, i.e. all class invariants (and the component invariant) are valid.