Object-Oriented Software Engineering

Using UML, Patterns, and Java

Mapping Models to Code

UML views & code generation

- **The architectural/structural view**
  - The EMF/Ecore approach - **Dave Steinberg, Frank Budinsky etc. - Eclipse Modeling Framework Second Edition, AWL 2009**
  - The UML approach (considered by almost all UML tools)
  - **Strengths points**: complete generation of the code describing the structure - still differentiate by associations management
  - **Weak points**: missing the code describing the behavior – **possible solution**: generating the code corresponding to OCL specifications (assertions & observers see: **OCLE**)
- **The behavioral view**
  - Code generation from state machines/state-charts diagrams – see **IBM Rational Rhapsody**
  - https://www.ibm.com/docs/en/SSB2MU_8.3.1/com.ibm.rhp.oem.pdf.doc/pdf/btc/Rhapsody%20Reference%20Workflow%20Guide%20IEC%2061508.pdf
  - Code generation from sequence/collaboration diagrams – mentioned in different academic papers

State of the Art: Model-based Software Engineering

- The Vision
  - During object design we build an object design model that realizes the use case model, and which is the basis for implementation (model-driven design)
- The Reality
  - Working on the object design model involves many activities that are error prone
  - Examples:
    - A new parameter must be added to an operation. Because of time pressure it is added to the source code, but not to the object model
    - Additional attributes are added to an entity object, but the data base table is not updated (as a result, the new attributes are not persistent).

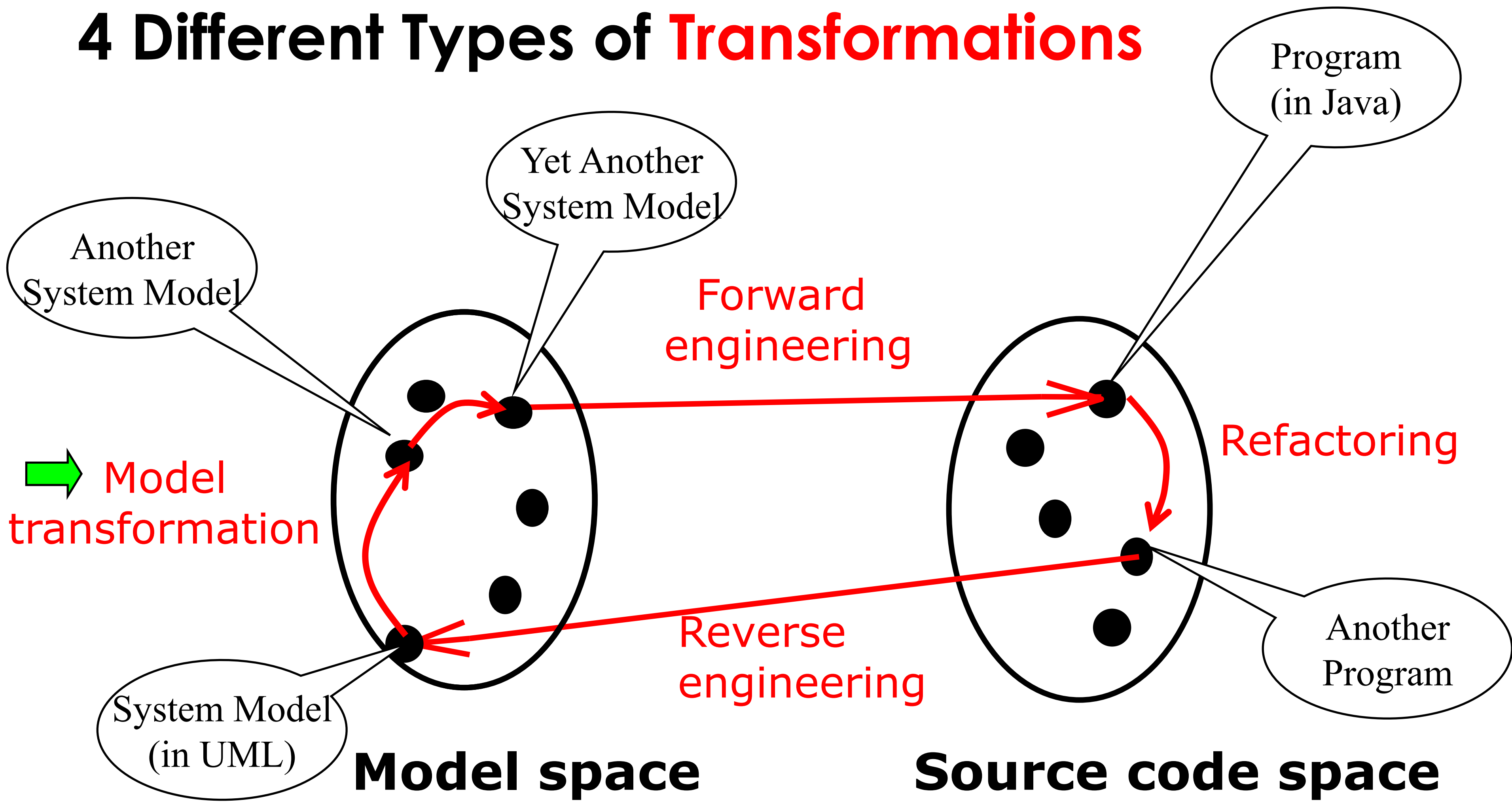
Other Object Design Activities

- Programming languages do not support the concept of a UML association
  - The associations of the object model must be transformed into collections of object references
- Many programming languages do not support contracts (invariants, pre and post conditions)
  - Developers must therefore manually transform contract specification into source code for detecting and handling contract violations
- The client changes the requirements during object design
  - The developer must change the interface specification of the involved classes
- All these object design activities cause problems, because they need to be done manually.

Handling the above-mentioned problems

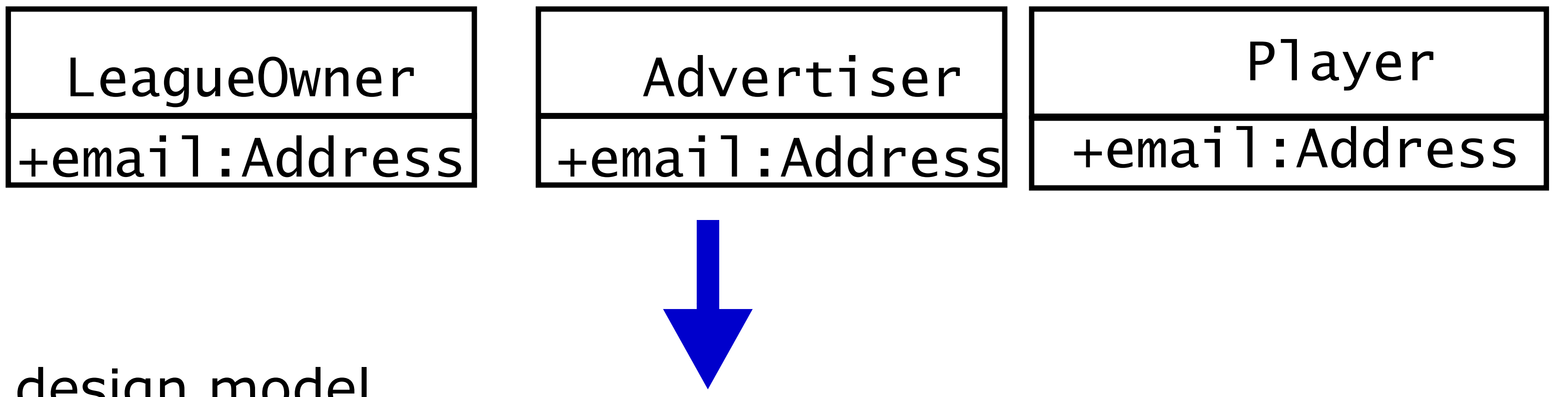
- Let us get a handle on these problems
- To do this we distinguish two kinds of spaces
  - the model space and the source code space
- and 4 different types of transformations
  - Model transformation
  - Forward engineering
  - Reverse engineering
  - Refactoring.

4 Different Types of Transformations

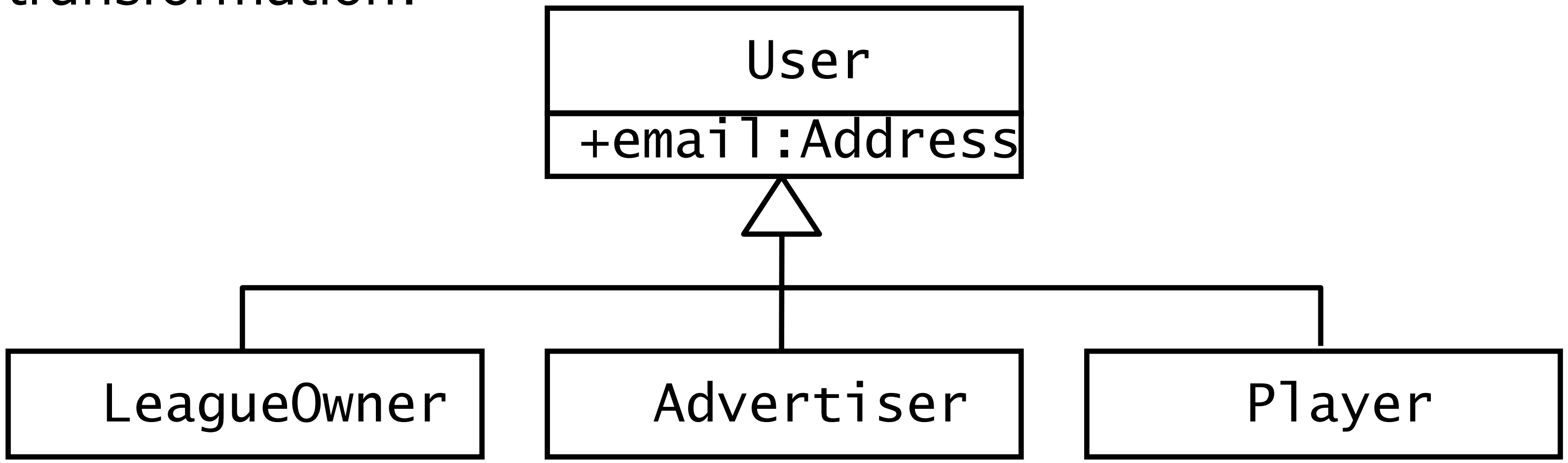


Model Transformation Example

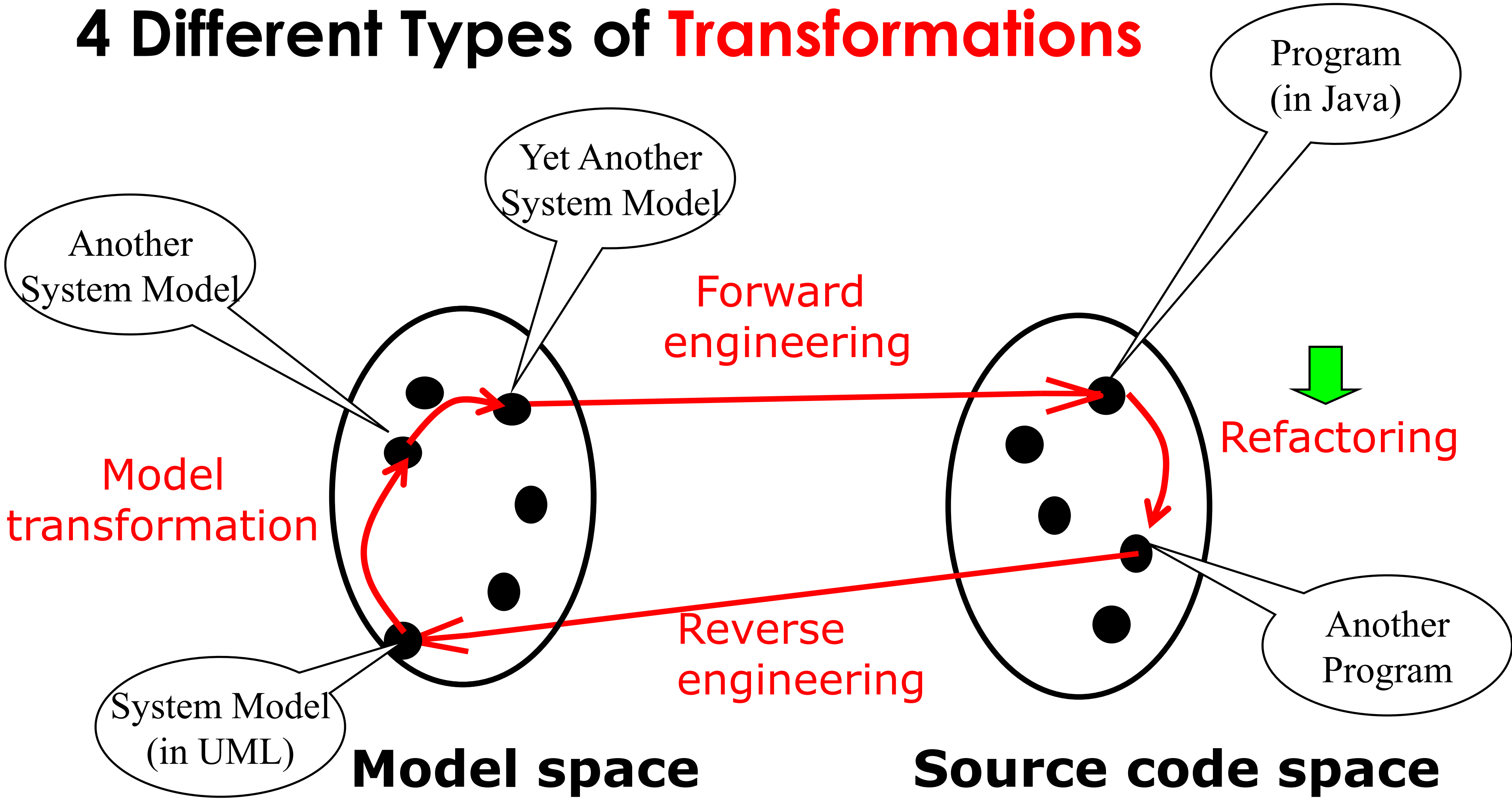
Object design model before transformation:



Object design model after transformation:



4 Different Types of Transformations



Refactoring Example: Pull Up Field

```
public class Player {
  private String email;
  //...
}
public class LeagueOwner {
  private String eMail;
  //...
}
public class Advertiser {
  private String
  email_address;
  //...
}
```

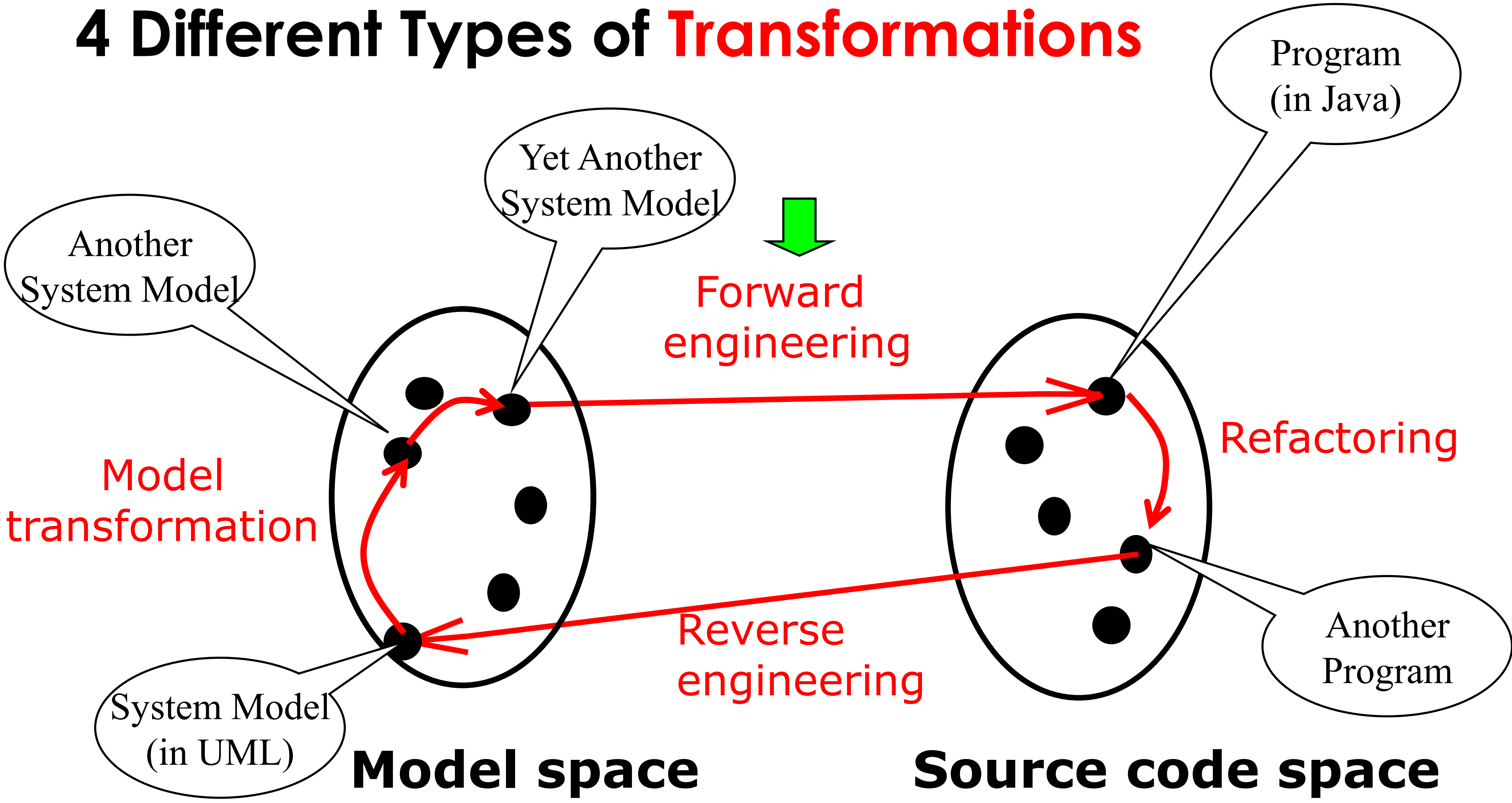
```
public class User {
  private String email;
}

public class Player extends User {
  //...
}

public class LeagueOwner extends
  User {
  //...
}

public class Advertiser extends
  User {
  //...
}
```

Refactoring Example: Pull Up Constructor Body

```
public class User {
   private String email;
}


public class Player extends User {
   public Player(String email) {
        this.email = email;
   }
}
public class LeagueOwner extends
   User{
   public LeagueOwner(String email) {
        this.email = email;
   }
}
public class Advertiser extendsUser{
   public Advertiser(String email) {
        this.email = email;
   }
}
```

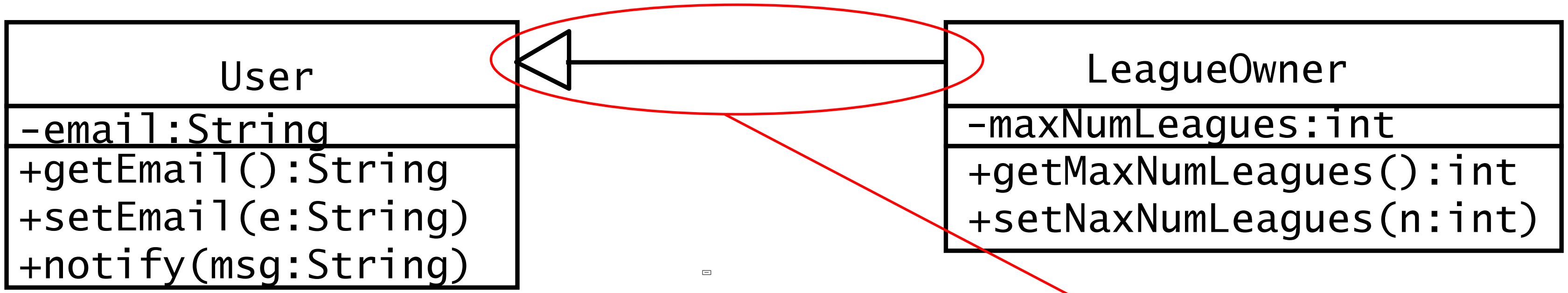
```
public class User {
   public User(String email) {
        this.email = email;
   }
}

public class Player extends User {
        public Player(String email)
{
                super(email);
        }
}
public class LeagueOwner extends
User {
        public LeagueOwner(String
email) {
                super(email);
        }
}
public class Advertiser extends Use
{
        public Advertiser(String
email) {
                super(email);
        }
}
```

4 Different Types of Transformations



Forward Engineering Example

Object design model before transformation:



Source code after transformation:

```
public class User {
   private String email;
   public String getEmail() {
         return email;
   }
   public void setEmail(String value){
         email = value;
   }
   public void notify(String msg) {
         // ....
   }
}
```

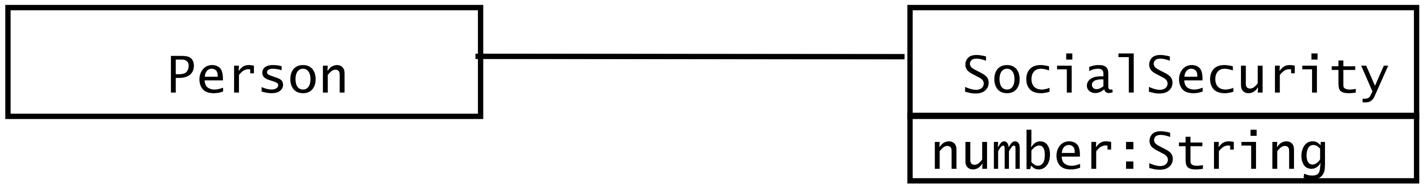
```
public class LeagueOwner extends User {
   private int maxNumLeagues;
   public int getMaxNumLeagues() {
         return maxNumLeagues;
   }
   public void setMaxNumLeagues
                  (int value) {
         maxNumLeagues = value;
   }
}
```

More Examples of Model Transformations and Forward Engineering

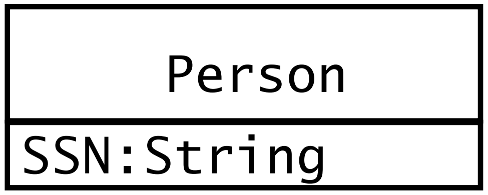
- Model Transformations
  - Goal: Optimizing the object design model
    - ➡ Collapsing objects
    - Delaying expensive computations
- Forward Engineering
  - Goal: Implementing the object design model in a programming language
  - Mapping inheritance
  - Mapping associations
  - Mapping contracts to exceptions
  - Mapping object models to tables

Collapsing Objects

Object design model before transformation:



Object design model after transformation:



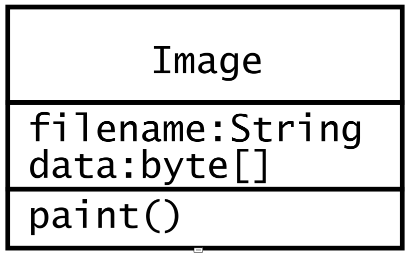
Turning an object into an attribute of another object is usually done, if the object does not have any interesting dynamic behavior (only get and set operations).

Examples of Model Transformations and Forward Engineering

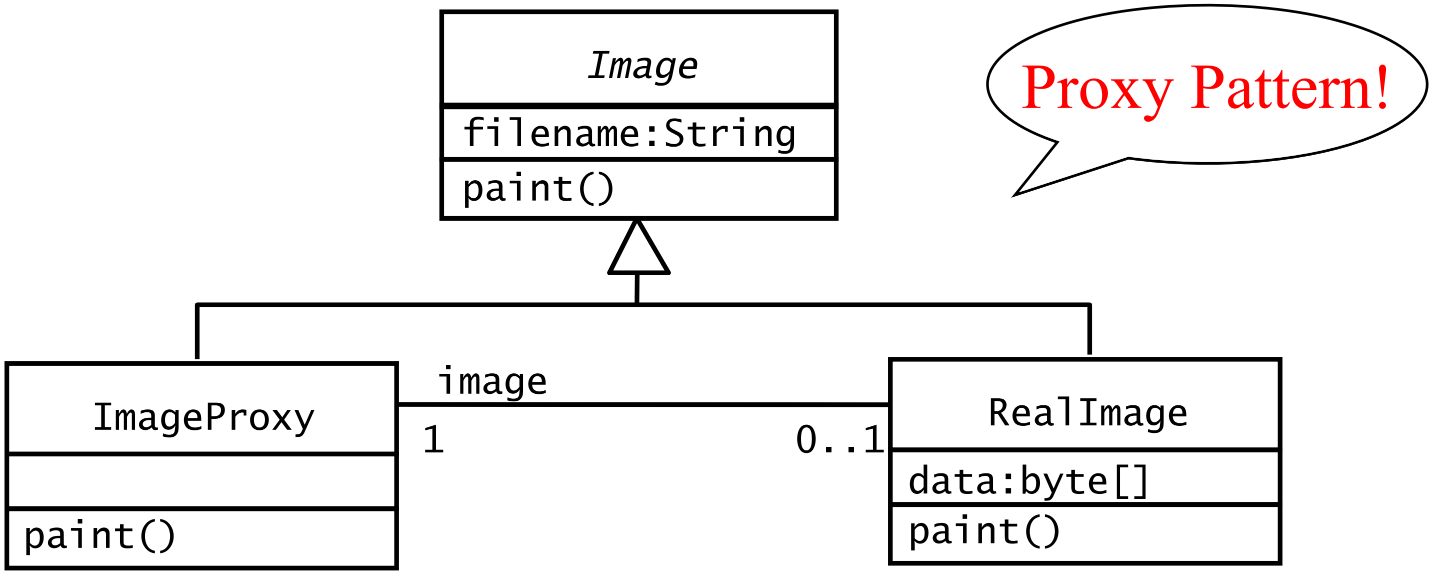
- Model Transformations
  - Goal: Optimizing the object design model
    - Collapsing objects
    - ➡ Delaying expensive computations
- Forward Engineering
  - Goal: Implementing the object design model in a programming language
  - Mapping inheritance
  - Mapping associations
  - Mapping contracts to exceptions
  - Mapping object models to tables

Delaying expensive computations

Object design model before transformation:



Object design model after transformation:



Using Different Views in Forward Engineering

- Architectural/Statical Views supports a complete automatic code generation 100%
  - Goal: Declaration of concepts: classes, interfaces, associations, implementation relationships (between classes and interfaces), import relationships (which correspond to dependencies in UML)
  - Full management of associations irrespective of their nature and properties, import declarations
- Dynamical Views
  - Goal: Generating the code corresponding to:
    - state transition diagrams and activities diagrams/flowcharts,
    - Sequence/Collaboration(Communication) diagram; method composition

Using Assertions and Observers in Forward Engineering

- OCL specifications can be transformed in OO programming languages. The management of assertion violation depends first on the target programming language and on the generated code, also.
  - In case of assertion violation, OCLE print a message.
- OCLE restrictions:
  - The current version does not consider code generation in case of Dynamical Views. However, due to support of Observers' code generation, the percent of code generated by OCLE is 70-75%. The missing part must be manually written.

Examples of Model Transformations and Forward Engineering

- Model Transformations
  - Goal: Optimizing the object design model
    - Collapsing objects
    - Delaying expensive computations
- Forward Engineering
  - Goal: Implementing the object design model in a programming language
  - ➡ Mapping inheritance
  - Mapping associations
  - Mapping contracts to exceptions
  - Mapping object models to tables

Forward Engineering: Mapping a UML Model into Source Code

- **Goal**: We have a UML-Model with inheritance. We want to translate it into source code
- **Question**: Which mechanisms in the programming language can be used?
  - Let’s focus on Java
- Java provides the following mechanisms:
  - Overwriting of methods (default in Java)
  - Final classes
  - Final methods
  - Abstract methods
  - Abstract classes
  - Interfaces.

Realizing Inheritance in Java

- Realisation of specialization and generalization
  - Definition of subclasses
  - Java keyword: **`extends`**
- Realisation of simple inheritance
  - Overwriting of methods is not allowed
  - Java keyword: **`final`**
- Realisation of implementation inheritance
  - Overwriting of methods
  - No keyword necessary:
    - Overwriting of methods is default in Java
- Realisation of specification inheritance
  - Specification of an interface
  - Java keywords: **`abstract`**, **`interface`**

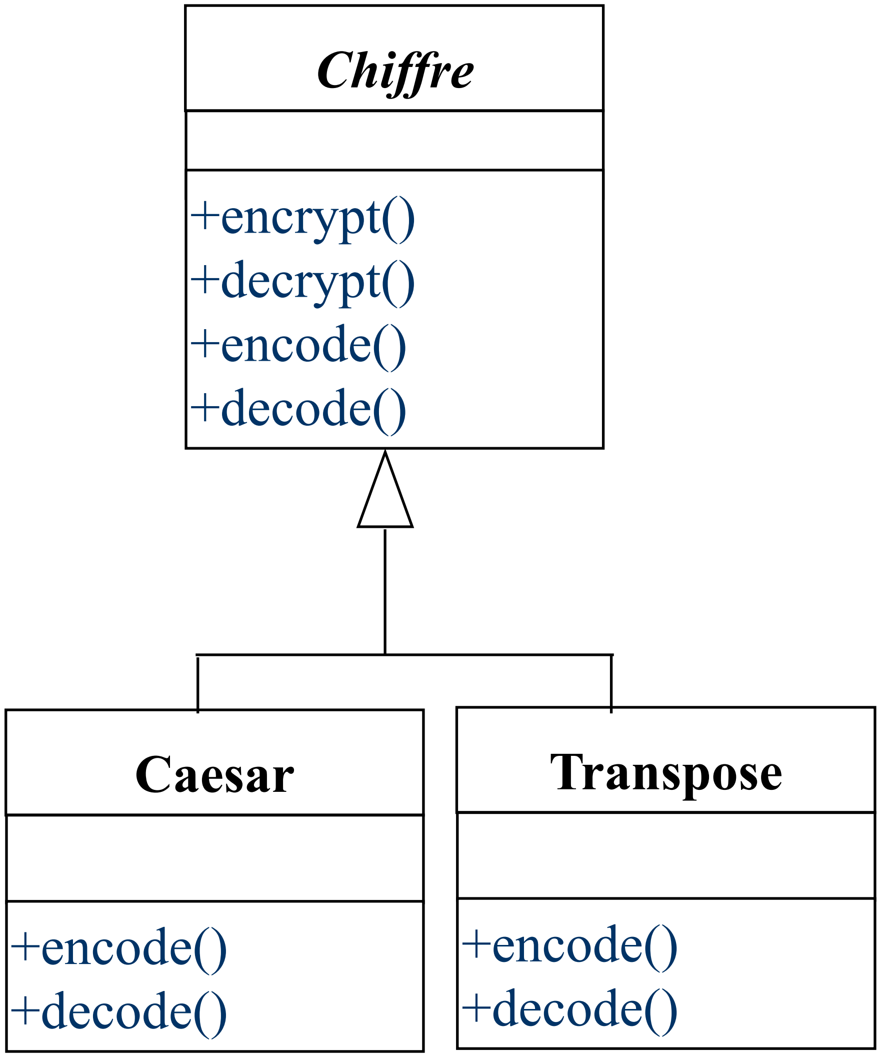
Example for the use of Abstract Methods: Cryptography

- Problem: Delivery a general encryption method
- Requirements:
  - The system provides algorithms for existing encryption methods (e.g. Caesar, Transposition)
  - New encryption algorithms, when they become available, can be linked into the program at runtime, without any need to recompile the program
  - The choice of the best encryption method can also be done at runtime.

Object Design of Chiffre

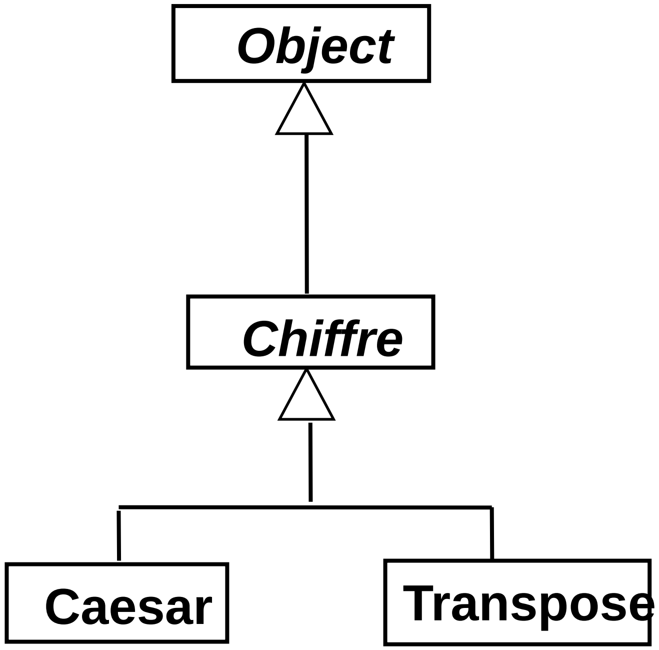
- We define a super class **`Chiffre`** and define subclasses for the existing existing encryption methods

- 4 public methods:
  - **`encrypt()`** encrypts a text of words
  - **`decrypt()`** deciphers a text of words
  - **`encode()`** uses a special algorithm for encryption of a single word
  - **`decode()`** uses a special algorithm for decryption of a single word.



Implementation of Chiffre in Java

- The methods `encrypt()` and `decrypt()` are the same for each subclass and can therefore be *implemented* in the superclass `Chiffre`
  - `Chiffre` is defined as subclass of `Object`, because we will use some methods of `Object`
- The methods `encode()` and `decode()` are specific for each subclass
  - We therefore define them as *abstract methods* in the super class and expect that they are *implemented* in the respective subclasses.

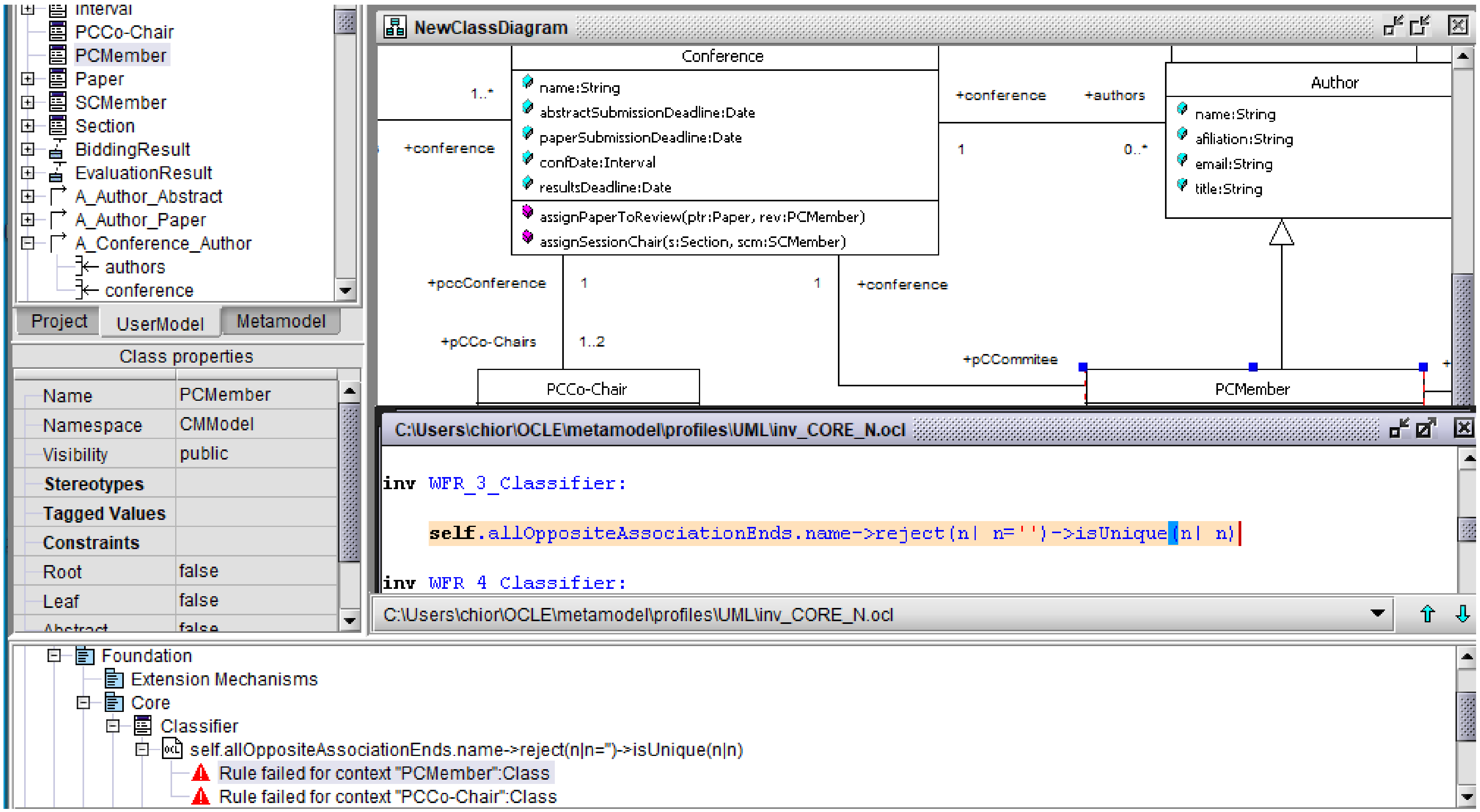


Exercise: Write the corresponding Java Code!

Examples of Model Transformations and Forward Engineering

- Model Transformations
  - Goal: Optimizing the object design model
    - ✓ Collapsing objects
    - ✓ Delaying expensive computations
- Forward Engineering
  - Goal: Implementing the object design model in a programming language
  - Is it the model compilable?
  - ✓ Mapping inheritance
  - Mapping associations
  - Mapping contracts to exceptions
  - Mapping object models to tables

Is it the model compilable?



Class properties

Name	PCMember
Namespace	CMModel
Visibility	public
Stereotypes	
Tagged Values	
Constraints	
Root	false
Leaf	false
Abstract	false

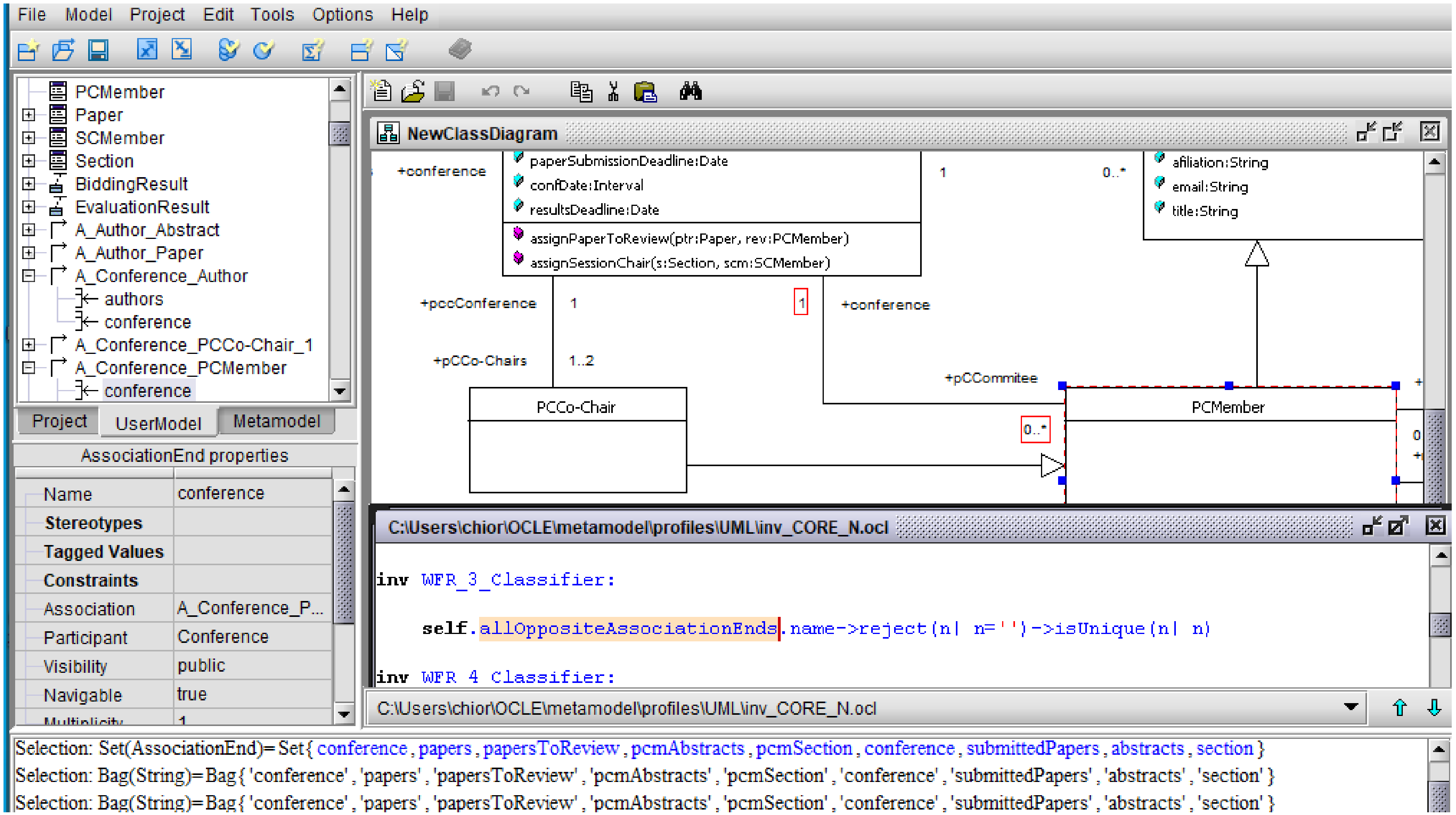
C:\Users\chior\OCLE\metamodel\profiles\UML\inv_CORE_N.ocl

```
inv WFR_3_Classifier:
    self.allOppositeAssociationEnds.name->reject(n| n='')->isUnique(n| n)

inv WFR_4_Classifier:
```

C:\Users\chior\OCLE\metamodel\profiles\UML\inv_CORE_N.ocl

```
Foundation
    Extension Mechanisms
    Core
        Classifier
            self.allOppositeAssociationEnds.name->reject(n|n='')->isUnique(n|n)
                Rule failed for context "PCMember":Class
                Rule failed for context "PCCo-Chair":Class
```

Is it the model compilable? cont.



AssociationEnd properties

Name	conference
Stereotypes	
Tagged Values	
Constraints	
Association	A_Conference_P...
Participant	Conference
Visibility	public
Navigable	true
Multiplicity	1

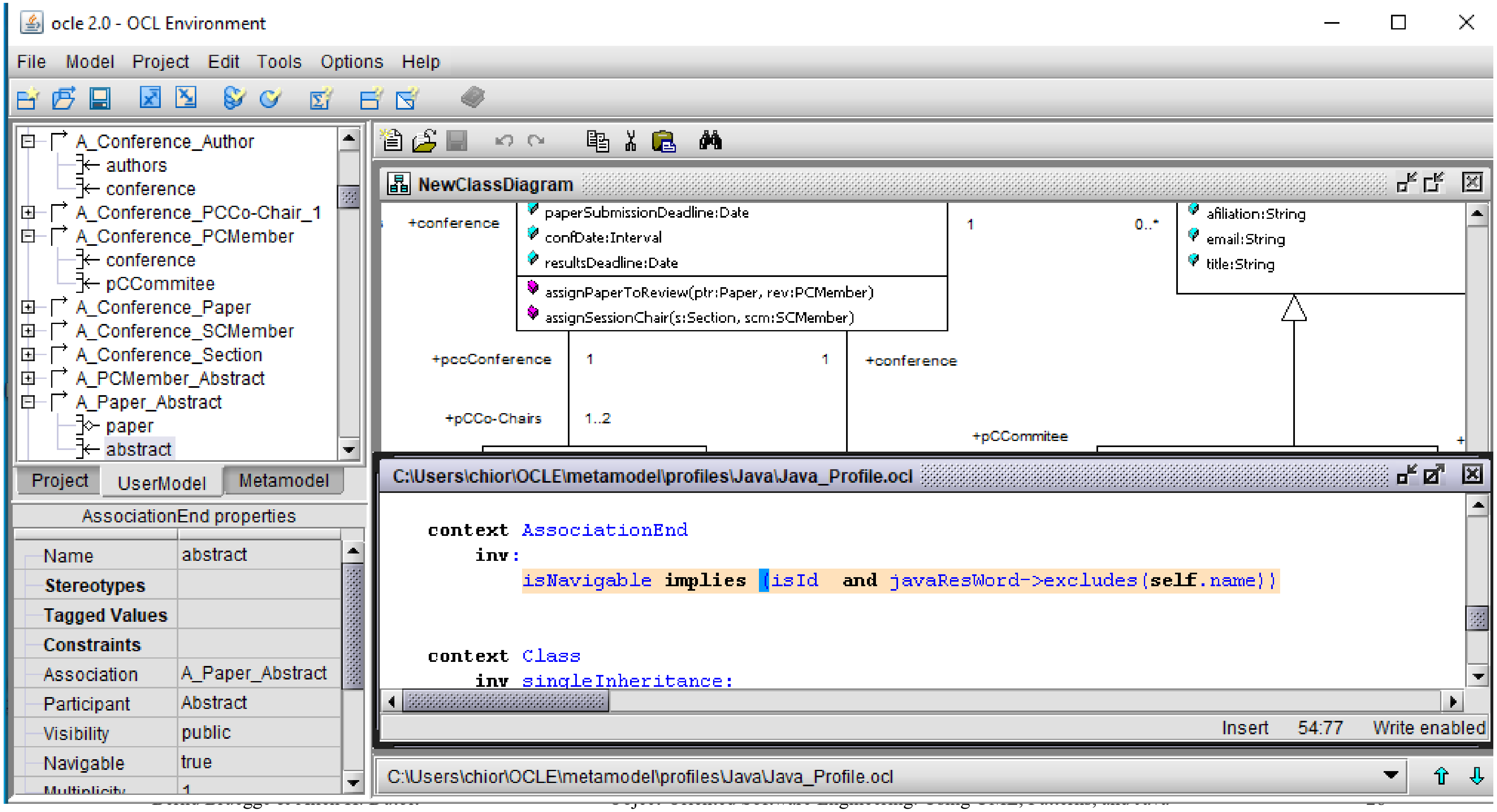
C:\Users\chior\OCLE\metamodel\profiles\UML\inv_CORE_N.ocl

```
inv WFR_3_Classifier:

    self.allOppositeAssociationEnds.name->reject(n| n='')->isUnique(n| n)

inv WFR_4_Classifier:
```

C:\Users\chior\OCLE\metamodel\profiles\UML\inv_CORE_N.ocl

```
Selection: Set(AssociationEnd)=Set{ conference , papers , papersToReview , pcmAbstracts , pcmSection , conference , submittedPapers , abstracts , section }
Selection: Bag(String)=Bag{ 'conference' , 'papers' , 'papersToReview' , 'pcmAbstracts' , 'pcmSection' , 'conference' , 'submittedPapers' , 'abstracts' , 'section' }
Selection: Bag(String)=Bag{ 'conference' , 'papers' , 'papersToReview' , 'pcmAbstracts' , 'pcmSection' , 'conference' , 'submittedPapers' , 'abstracts' , 'section' }
```

Is it the model compilable? Java Profile


ocle 2.0 - OCL Environment
File Model Project Edit Tools Options Help
NewClassDiagram
Project UserModel Metamodel
AssociationEnd properties
Insert 54:77 Write enabled

```
A_Conference_Author
    authors
    conference
A_Conference_PCCo-Chair_1
A_Conference_PCMember
    conference
    pCCommitee
A_Conference_Paper
A_Conference_SCMember
A_Conference_Section
A_PCMember_Abstract
A_Paper_Abstract
    paper
    abstract
```

Name	abstract
Stereotypes	
Tagged Values	
Constraints	
Association	A_Paper_Abstract
Participant	Abstract
Visibility	public
Navigable	true
Multiplicity	1

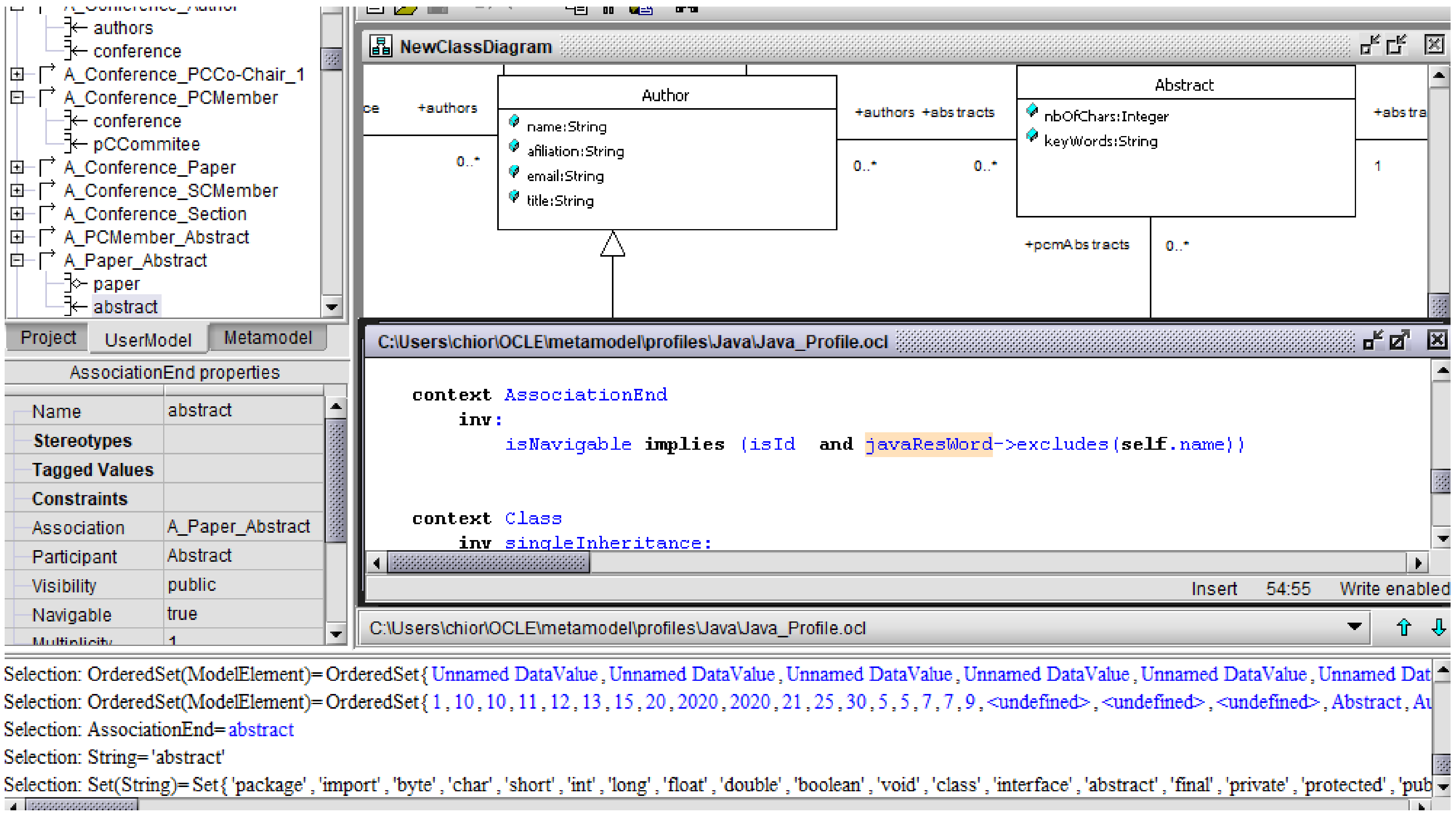
```
+conference
paperSubmissionDeadline:Date
confDate:Interval
resultsDeadline:Date
assignPaperToReview(ptr:Paper, rev:PCMember)
assignSessionChair(s:Section, scm:SCMember)
1    0..*
afiliation:String
email:String
title:String
+pccConference 1    1 +conference
+pCCo-Chairs 1..2
+pCCommitee
```

C:\Users\chior\OCLE\metamodel\profiles\Java\Java_Profile.ocl

```
context AssociationEnd
    inv:
        isNavigable implies (isId  and javaResWord->excludes(self.name))


context Class
    inv singleInheritance:
```

C:\Users\chior\OCLE\metamodel\profiles\Java\Java_Profile.ocl

Is it the model compilable? Java Profile



AssociationEnd properties

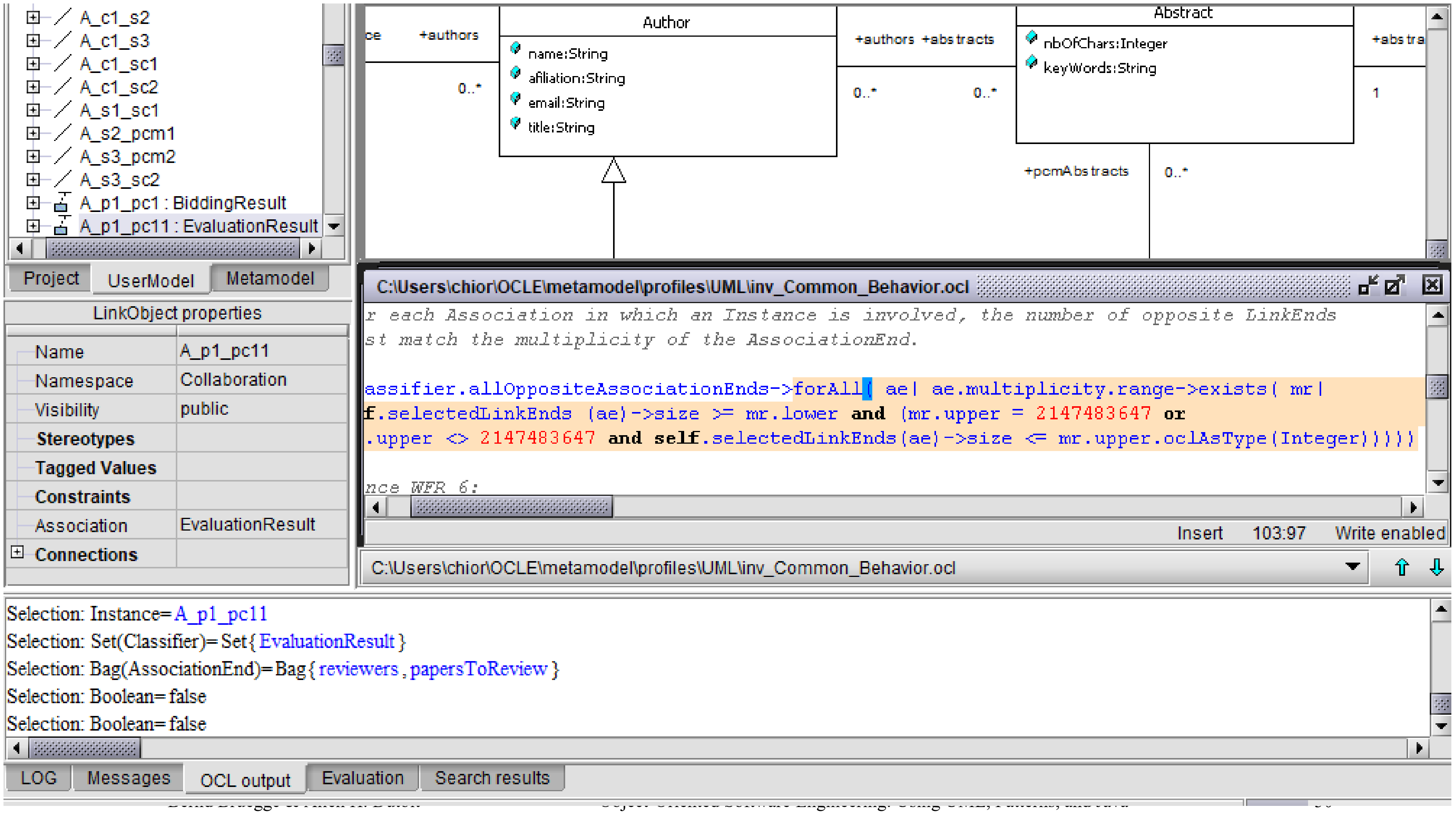
Name	abstract
Stereotypes	
Tagged Values	
Constraints	
Association	A_Paper_Abstract
Participant	Abstract
Visibility	public
Navigable	true
Multiplicity	1

C:\Users\chior\OCLE\metamodel\profiles\Java\Java_Profile.ocl

```
context AssociationEnd
    inv:
        isNavigable implies (isId  and javaResWord->excludes(self.name))


context Class
    inv singleInheritance:
```

```
Selection: OrderedSet(ModelElement)=OrderedSet{ Unnamed DataValue , Unnamed DataValue , Unnamed DataValue , Unnamed DataValue , Unnamed DataValue , Unnamed Dat
Selection: OrderedSet(ModelElement)=OrderedSet{ 1 , 10 , 10 , 11 , 12 , 13 , 15 , 20 , 2020 , 2020 , 21 , 25 , 30 , 5 , 5 , 7 , 7 , 9 , <undefined> , <undefined> , <undefined> , Abstract , Au
Selection: AssociationEnd= abstract
Selection: String='abstract'
Selection: Set(String)=Set{ 'package' , 'import' , 'byte' , 'char' , 'short' , 'int' , 'long' , 'float' , 'double' , 'boolean' , 'void' , 'class' , 'interface' , 'abstract' , 'final' , 'private' , 'protected' , 'pub
```

Is it the model compilable? Common behavior



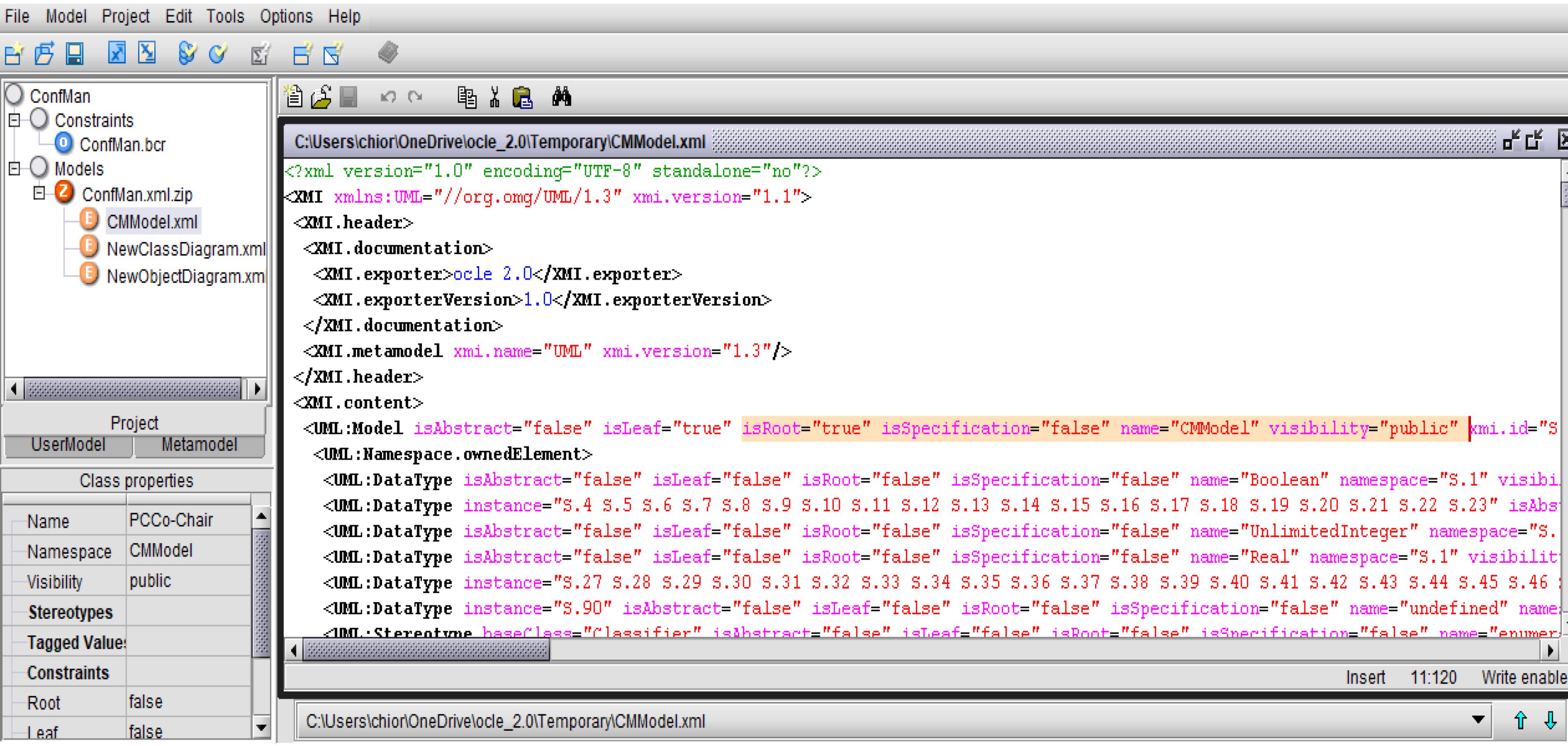
LinkObject properties

Name	A_p1_pc11
Namespace	Collaboration
Visibility	public
Stereotypes	
Tagged Values	
Constraints	
Association	EvaluationResult
Connections	

```
r each Association in which an Instance is involved, the number of opposite LinkEnds
st match the multiplicity of the AssociationEnd.

assifier.allOppositeAssociationEnds->forAll( ae| ae.multiplicity.range->exists( mr|
f.selectedLinkEnds (ae)->size >= mr.lower and (mr.upper = 2147483647 or
.upper <> 2147483647 and self.selectedLinkEnds(ae)->size <= mr.upper.oclAsType(Integer))))

nce WFR 6:
```

```
Selection: Instance=A_p1_pc11
Selection: Set(Classifier)=Set{ EvaluationResult }
Selection: Bag(AssociationEnd)=Bag{ reviewers , papersToReview }
Selection: Boolean= false
Selection: Boolean= false
```

Model serialization in OCLE - CMModel



C:\Users\chior\OneDrive\ocle_2.0\Temporary\CMModel.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<XMI xmlns:UML="//org.omg/UML/1.3" xmi.version="1.1">
 <XMI.header>
  <XMI.documentation>
   <XMI.exporter>ocle 2.0</XMI.exporter>
   <XMI.exporterVersion>1.0</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
 </XMI.header>
 <XMI.content>
  <UML:Model isAbstract="false" isLeaf="true" isRoot="true" isSpecification="false" name="CMModel" visibility="public" xmi.id="S
   <UML:Namespace.ownedElement>
    <UML:DataType isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="Boolean" namespace="S.1" visibi
    <UML:DataType instance="S.4 S.5 S.6 S.7 S.8 S.9 S.10 S.11 S.12 S.13 S.14 S.15 S.16 S.17 S.18 S.19 S.20 S.21 S.22 S.23" isAbs
    <UML:DataType isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="UnlimitedInteger" namespace="S.
    <UML:DataType isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="Real" namespace="S.1" visibilit
    <UML:DataType instance="S.27 S.28 S.29 S.30 S.31 S.32 S.33 S.34 S.35 S.36 S.37 S.38 S.39 S.40 S.41 S.42 S.43 S.44 S.45 S.46
    <UML:DataType instance="S.90" isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="undefined" name
    <UML:Stereotype baseClass="Classifier" isAbstract="false" isLeaf="false" isRoot="false" isSpecification="false" name="enumer
```

C:\Users\chior\OneDrive\ocle_2.0\Temporary\CMModel.xml

Name	PCCo-Chair
Namespace	CMModel
Visibility	public
Stereotypes	
Tagged Value	
Constraints	
Root	false
Leaf	false

Velocity Template Engine 1.3rc1

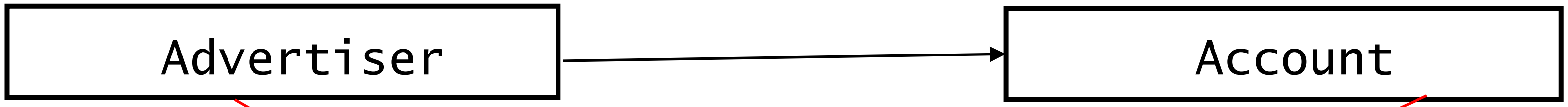
```
## Velocity template file for a public class declaration
#* Keys:
*       classname - the name of the class, not qualified
*       packagename - the qualified name of the package where the class is declared, such as ro.ubbcluj.lci.utils
*       importstatements - the import statements list required by the class
*       modifiers - the list of modifiers applied to the class; this must always include the "class" or "interface" modifier
* but not both simultaneously
*       extclasses - the list of clases extended by the class; each class is specified using its name, which may be qualified
*       implinterfaces - the list of interfaces implemented by the class; each interface is specified using its name, which
* may be qualified
*#
/*
 * @(#)${classname}.java
 *
 * Generated by <a href="http://lci.cs.ubbcluj.ro/ocle/>OCLE 2.0</a>
 * using <a href="http://jakarta.apache.org/velocity/">
 * Velocity Template Engine 1.3rc1</a>
 */
#if (${packagename.length()} > 0)package ${packagename};
#end
#import_list(${importstatements})

/**
 *
 * @author unascribed
 */
#list(" " ${modifiers}) ${classname}#if (${extclasses.size()} > 0) extends #argument_list(${extclasses})
#end#if (${implinterfaces.size()} > 0)
        implements #argument_list(${implinterfaces})#end#opening_brace()
```

Mapping Associations

1. Unidirectional one-to-one association
2. Bidirectional one-to-one association
3. Bidirectional one-to-many association
4. Bidirectional many-to-many association
5. Bidirectional qualified association.

Unidirectional one-to-one association

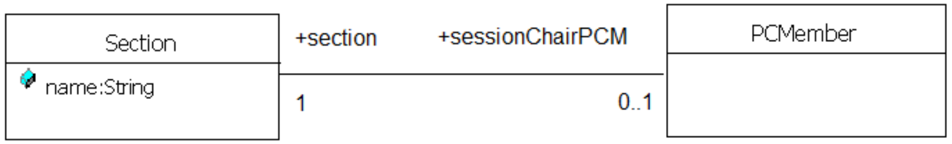
Object design model before transformation:

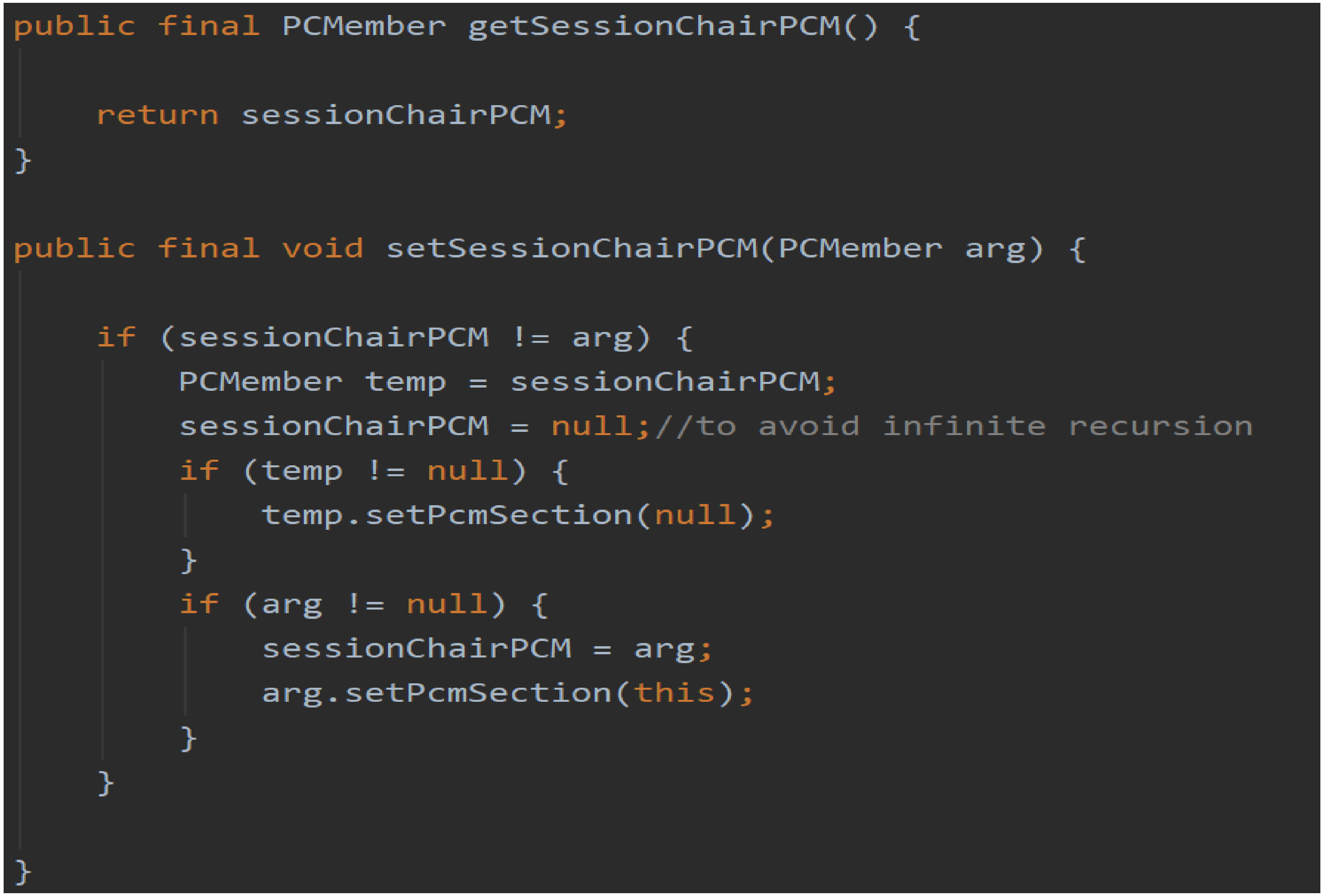


Source code after transformation:

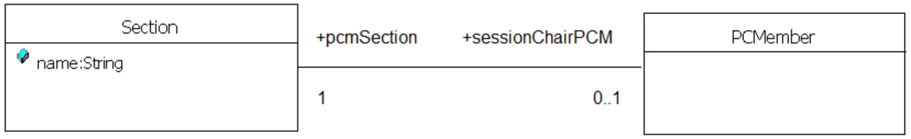
```
public class Advertiser {
      private Account account;
      public Advertiser() {
            account = new Account();
      }
      public Account getAccount() {
            return account;
      }
}
```

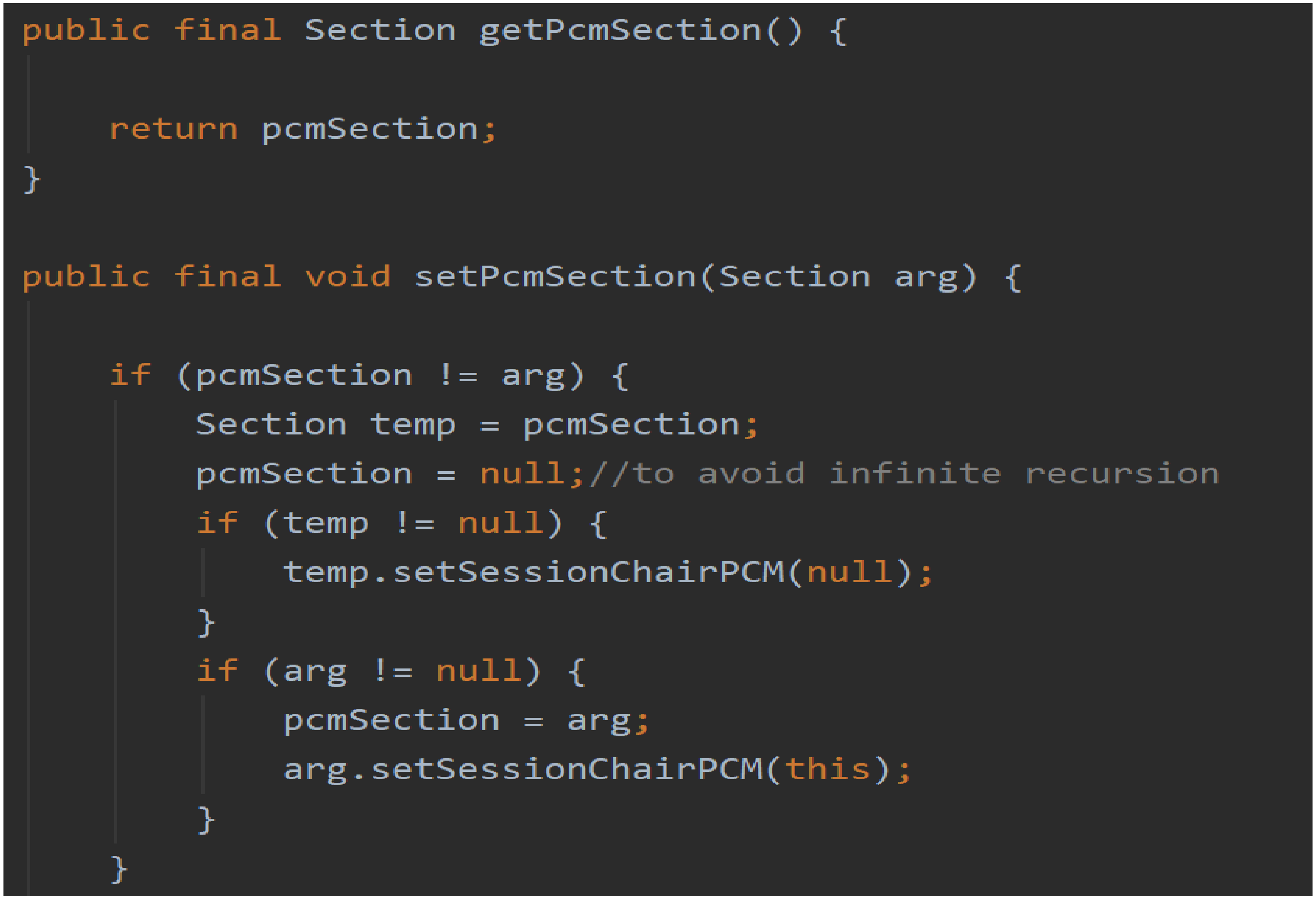
Bidirectional one-to-one association




```
public final PCMember getSessionChairPCM() {

    return sessionChairPCM;
}

public final void setSessionChairPCM(PCMember arg) {

    if (sessionChairPCM != arg) {
        PCMember temp = sessionChairPCM;
        sessionChairPCM = null;//to avoid infinite recursion
        if (temp != null) {
            temp.setPcmSection(null);
        }
        if (arg != null) {
            sessionChairPCM = arg;
            arg.setPcmSection(this);
        }
    }

}
```

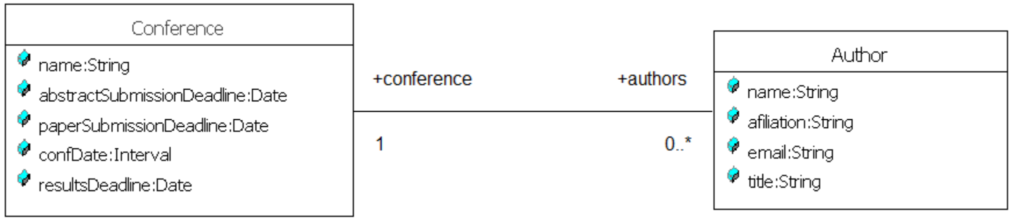
Bidirectional one-to-one association

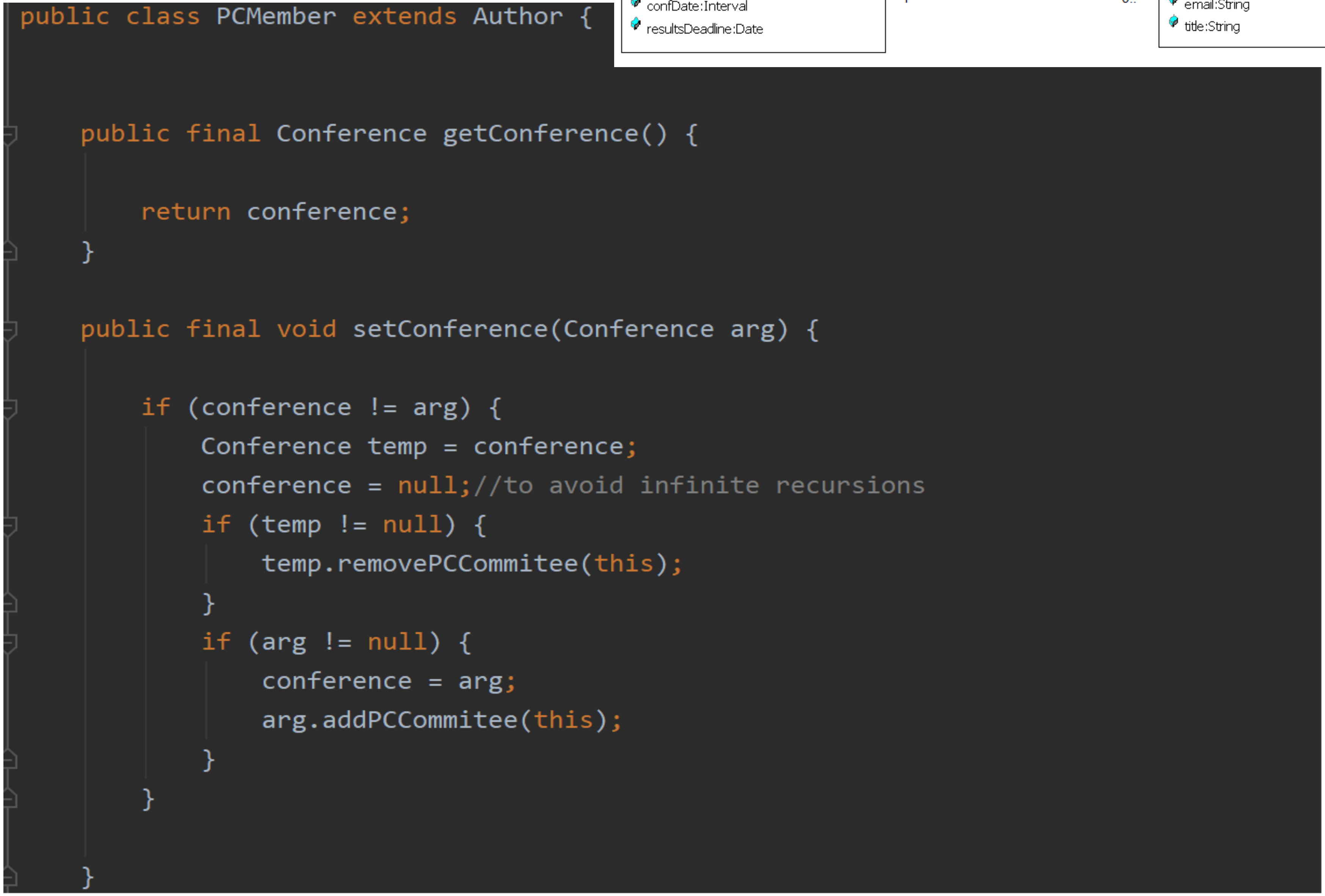




```
public final Section getPcmSection() {

    return pcmSection;
}

public final void setPcmSection(Section arg) {

    if (pcmSection != arg) {
        Section temp = pcmSection;
        pcmSection = null;//to avoid infinite recursion
        if (temp != null) {
            temp.setSessionChairPCM(null);
        }
        if (arg != null) {
            pcmSection = arg;
            arg.setSessionChairPCM(this);
        }
    }
```

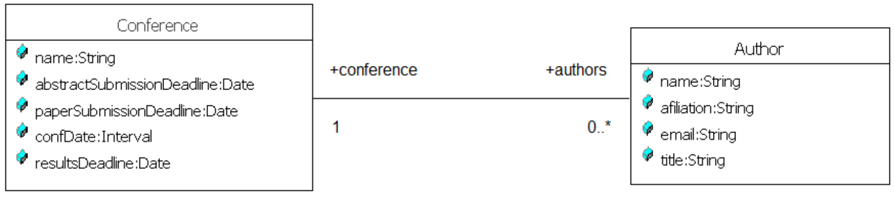
Bidirectional one-to-many association

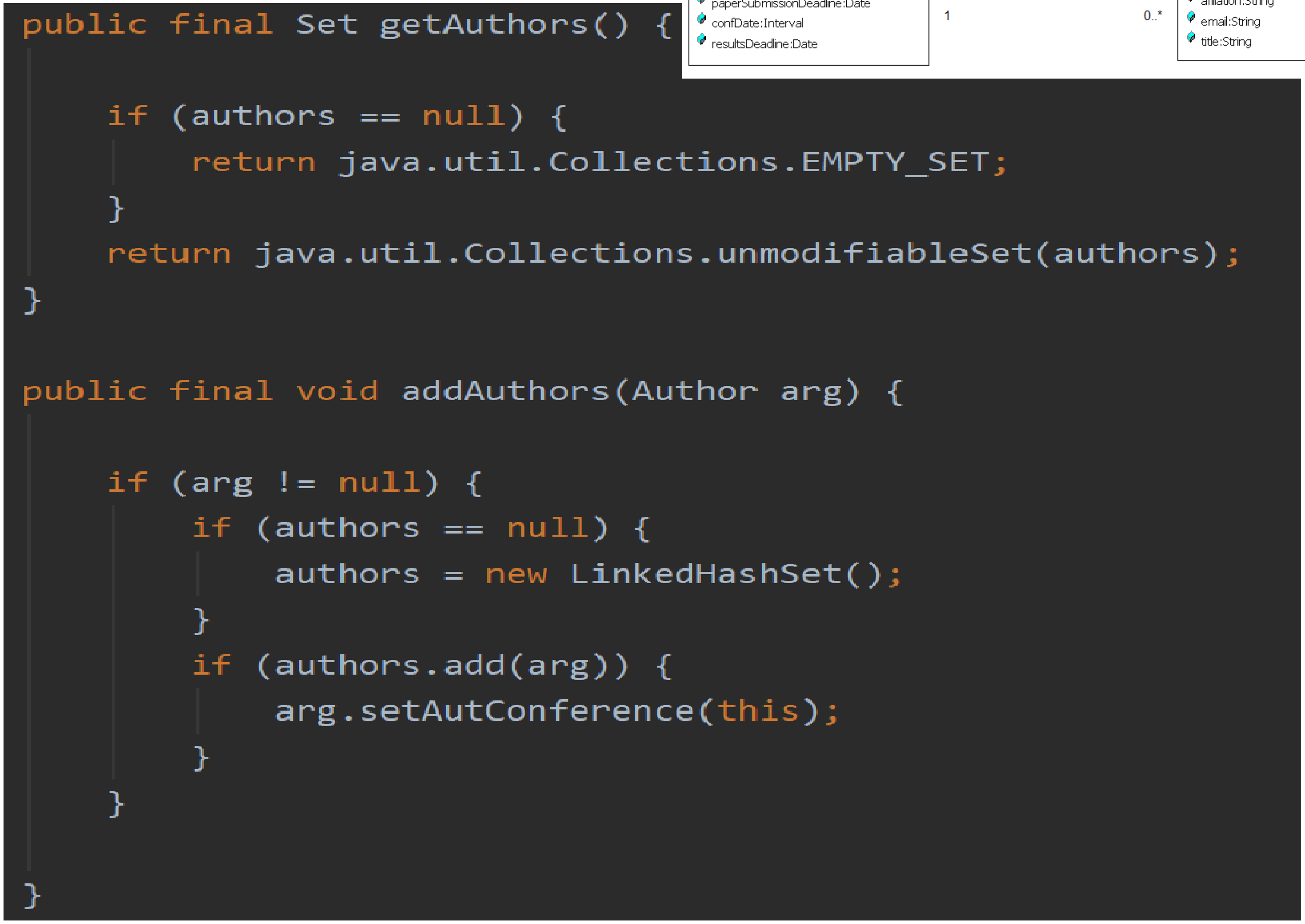




```
public class PCMember extends Author {


    public final Conference getConference() {

        return conference;
    }

    public final void setConference(Conference arg) {

        if (conference != arg) {
            Conference temp = conference;
            conference = null;//to avoid infinite recursions
            if (temp != null) {
                temp.removePCCommitee(this);
            }
            if (arg != null) {
                conference = arg;
                arg.addPCCommitee(this);
            }
        }

    }
```

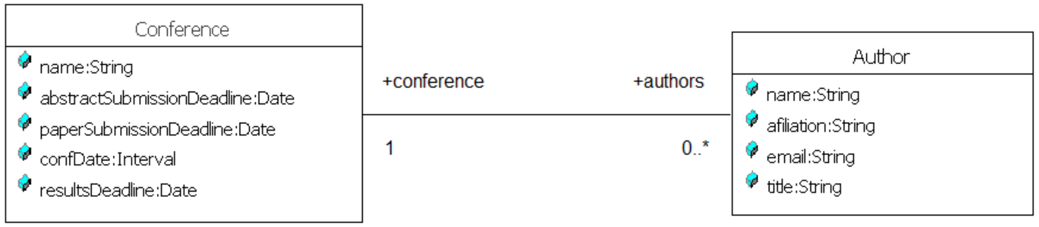
Bidirectional one-to-many association

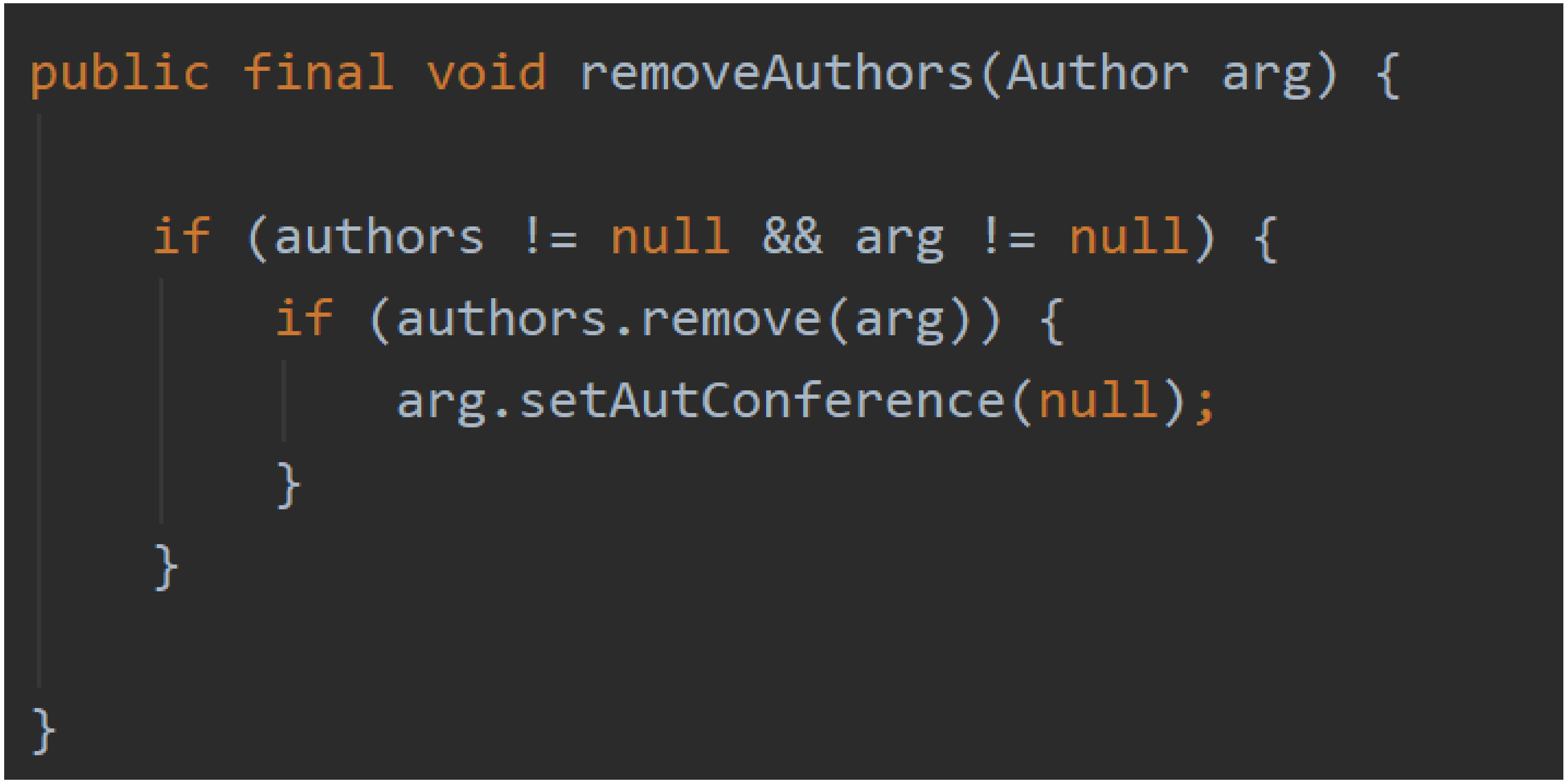




```java
public final Set getAuthors() {

    if (authors == null) {
        return java.util.Collections.EMPTY_SET;
    }
    return java.util.Collections.unmodifiableSet(authors);
}

public final void addAuthors(Author arg) {

    if (arg != null) {
        if (authors == null) {
            authors = new LinkedHashSet();
        }
        if (authors.add(arg)) {
            arg.setAutConference(this);
        }
    }

}
```

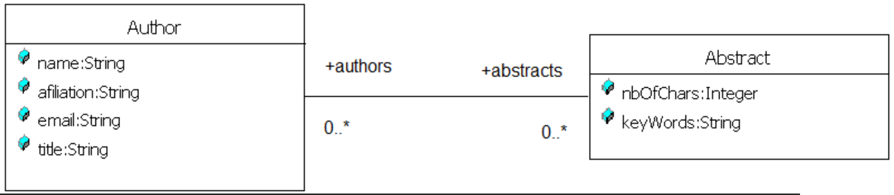
Bidirectional one-to-many association

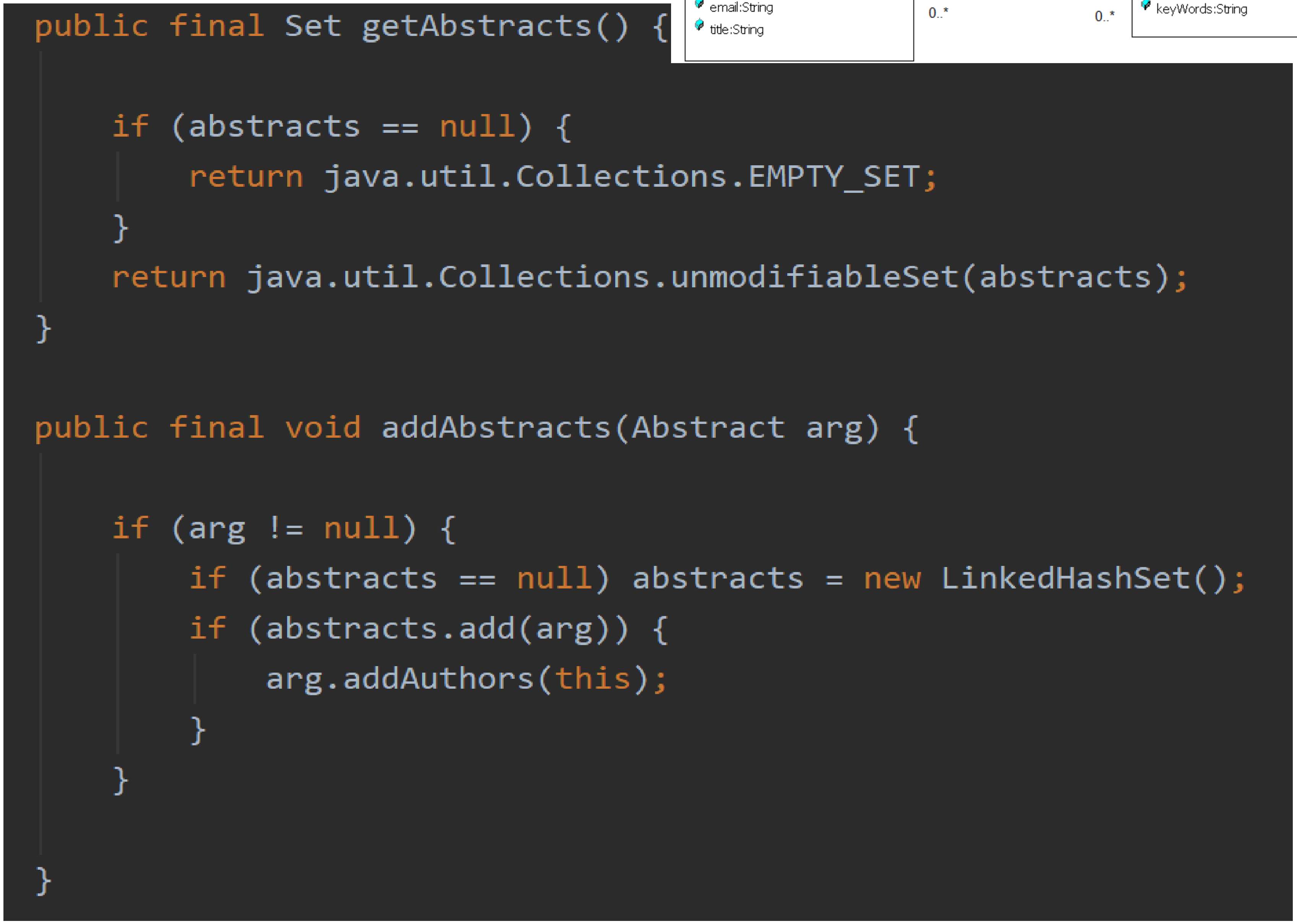




```
public final void removeAuthors(Author arg) {

    if (authors != null && arg != null) {
        if (authors.remove(arg)) {
            arg.setAutConference(null);
        }
    }

}
```

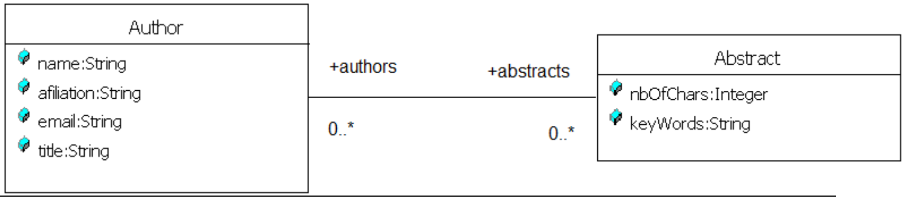
Bidirectional many-to-many association

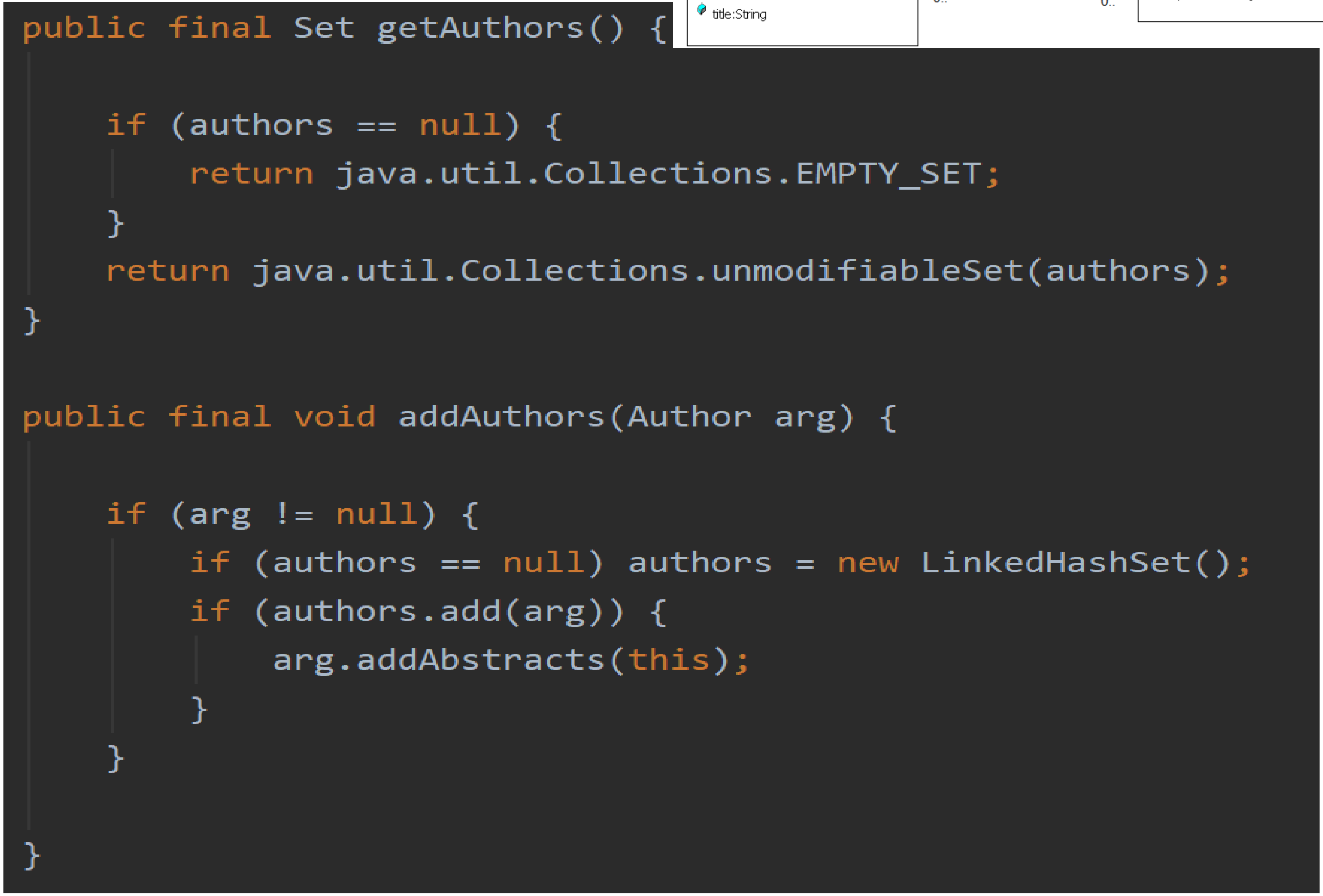




```
public final Set getAbstracts() {

    if (abstracts == null) {
        return java.util.Collections.EMPTY_SET;
    }
    return java.util.Collections.unmodifiableSet(abstracts);
}

public final void addAbstracts(Abstract arg) {

    if (arg != null) {
        if (abstracts == null) abstracts = new LinkedHashSet();
        if (abstracts.add(arg)) {
            arg.addAuthors(this);
        }
    }

}
```

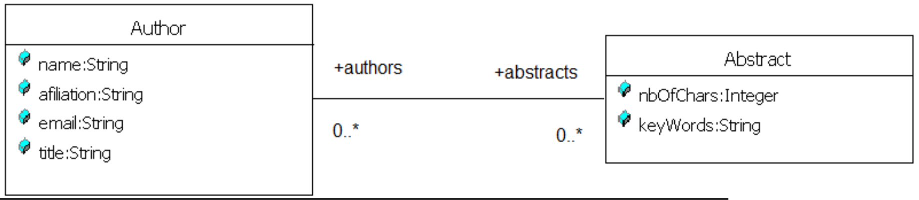
Bidirectional many-to-many association

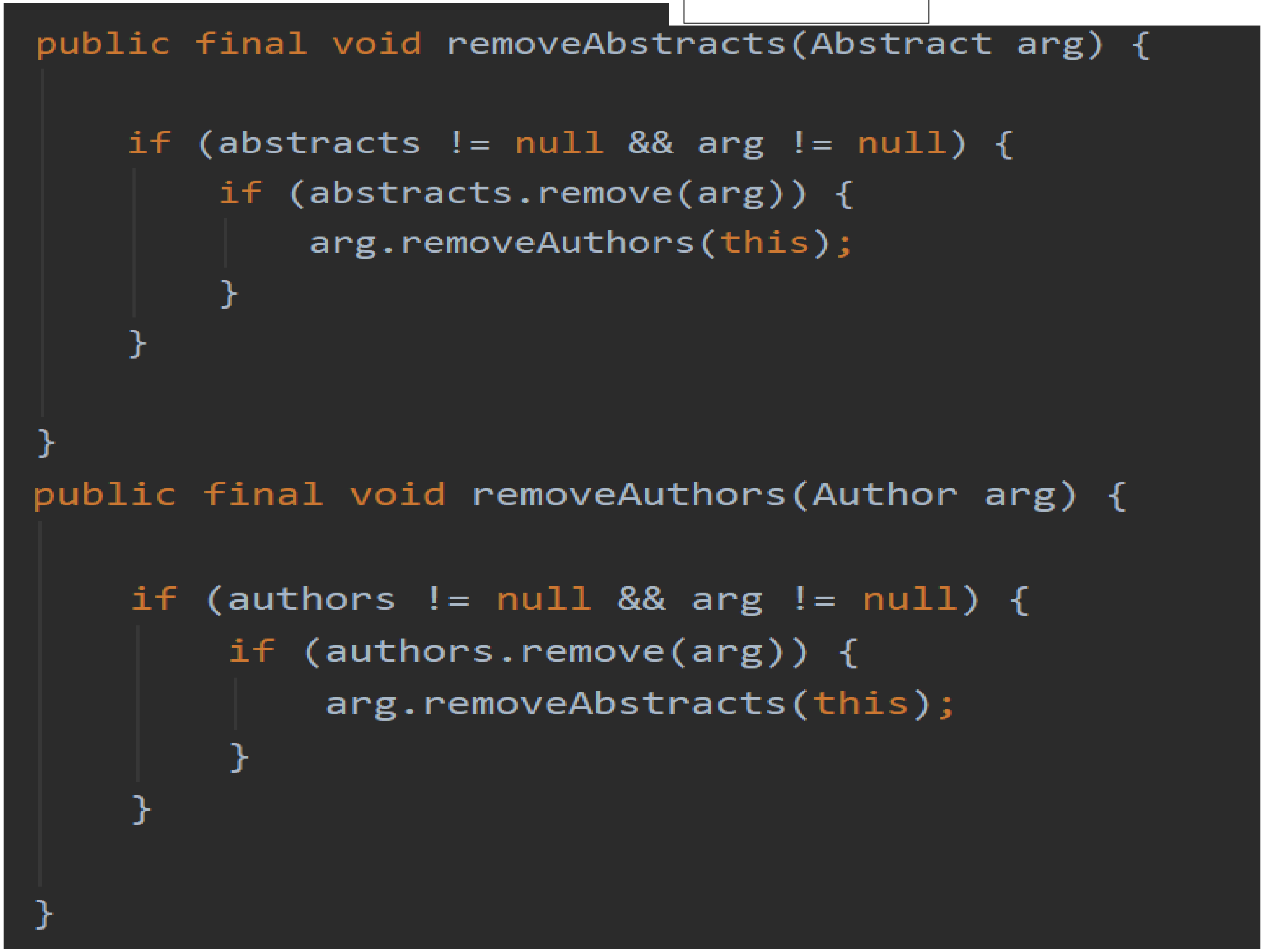




```java
public final Set getAuthors() {

    if (authors == null) {
        return java.util.Collections.EMPTY_SET;
    }
    return java.util.Collections.unmodifiableSet(authors);
}

public final void addAuthors(Author arg) {

    if (arg != null) {
        if (authors == null) authors = new LinkedHashSet();
        if (authors.add(arg)) {
            arg.addAbstracts(this);
        }
    }

}
```

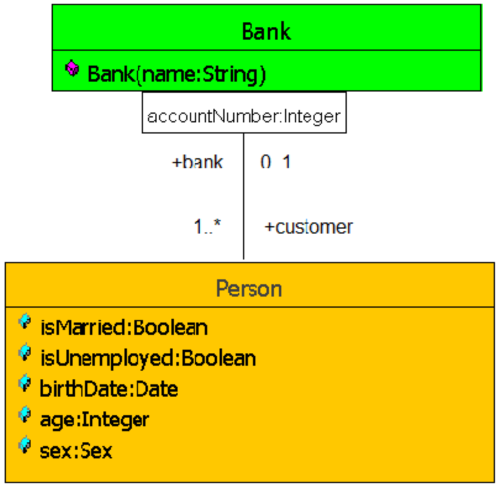
Bidirectional many-to-many association

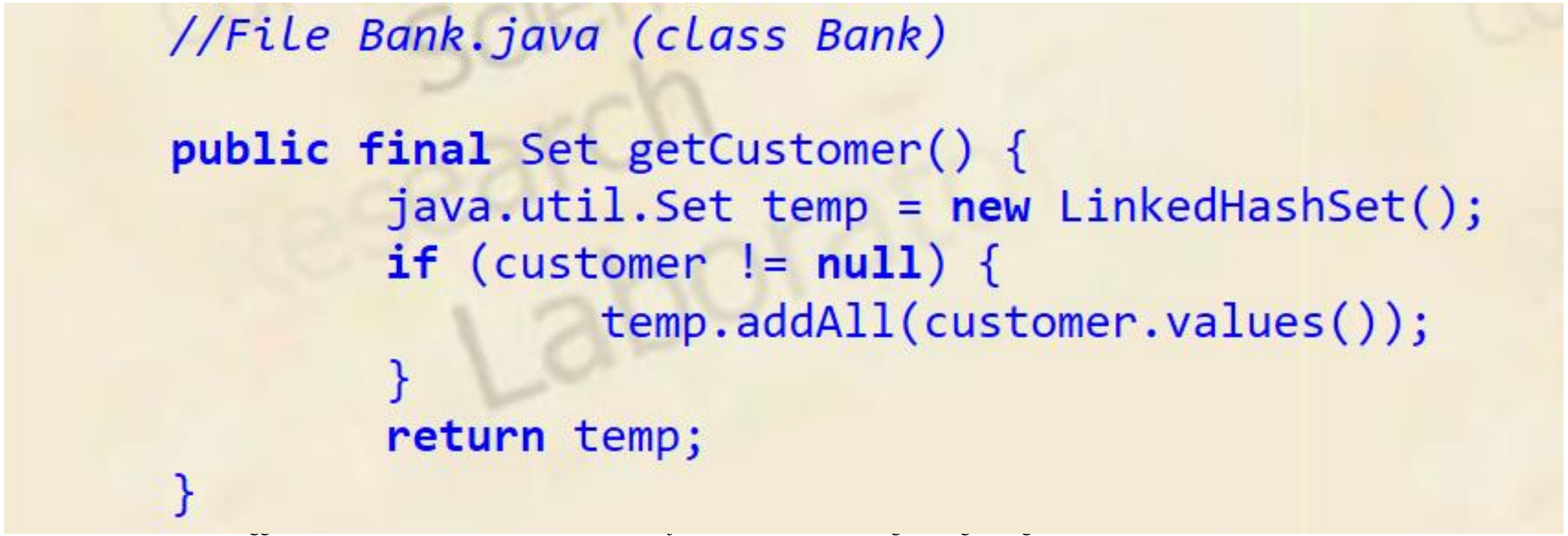




```
public final void removeAbstracts(Abstract arg) {

    if (abstracts != null && arg != null) {
        if (abstracts.remove(arg)) {
            arg.removeAuthors(this);
        }
    }

}
public final void removeAuthors(Author arg) {

    if (authors != null && arg != null) {
        if (authors.remove(arg)) {
            arg.removeAbstracts(this);
        }
    }

}
```

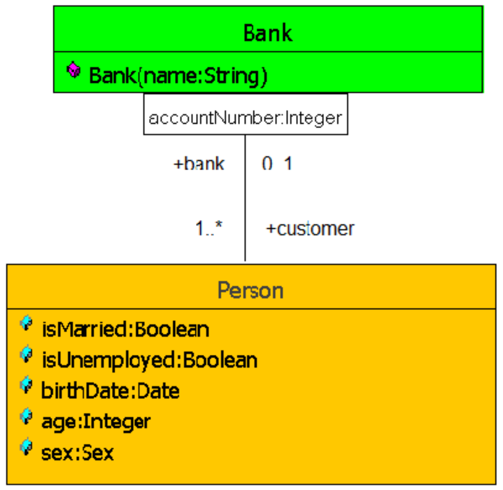
Bidirectional qualified association





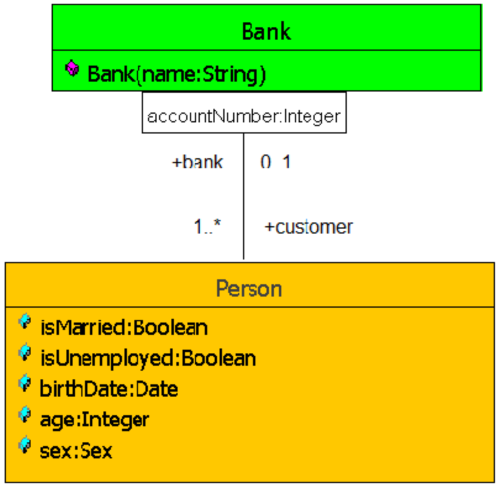
```
//File Bank.java (class Bank)

public final Set getCustomer() {
        java.util.Set temp = new LinkedHashSet();
        if (customer != null) {
                temp.addAll(customer.values());
        }
        return temp;
}
```

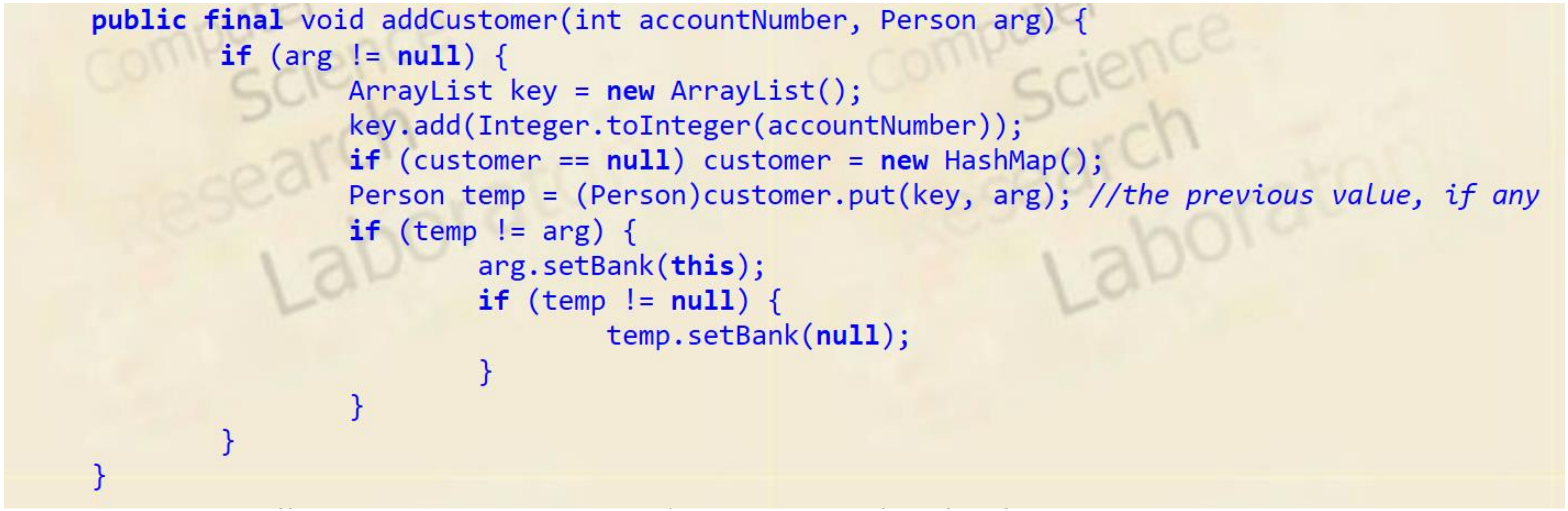
Bidirectional qualified association



```
    public final Person getCustomer(int accountNumber) {
            if (customer == null) return null;
            ArrayList key = new ArrayList();
            key.add(Integer.toInteger(accountNumber));
            return (Person)customer.get(key);
    }
```

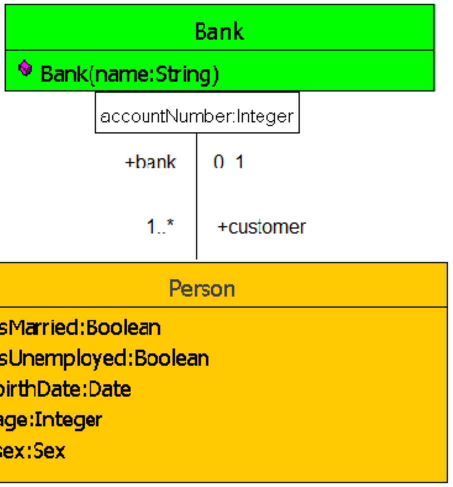
Bidirectional qualified association





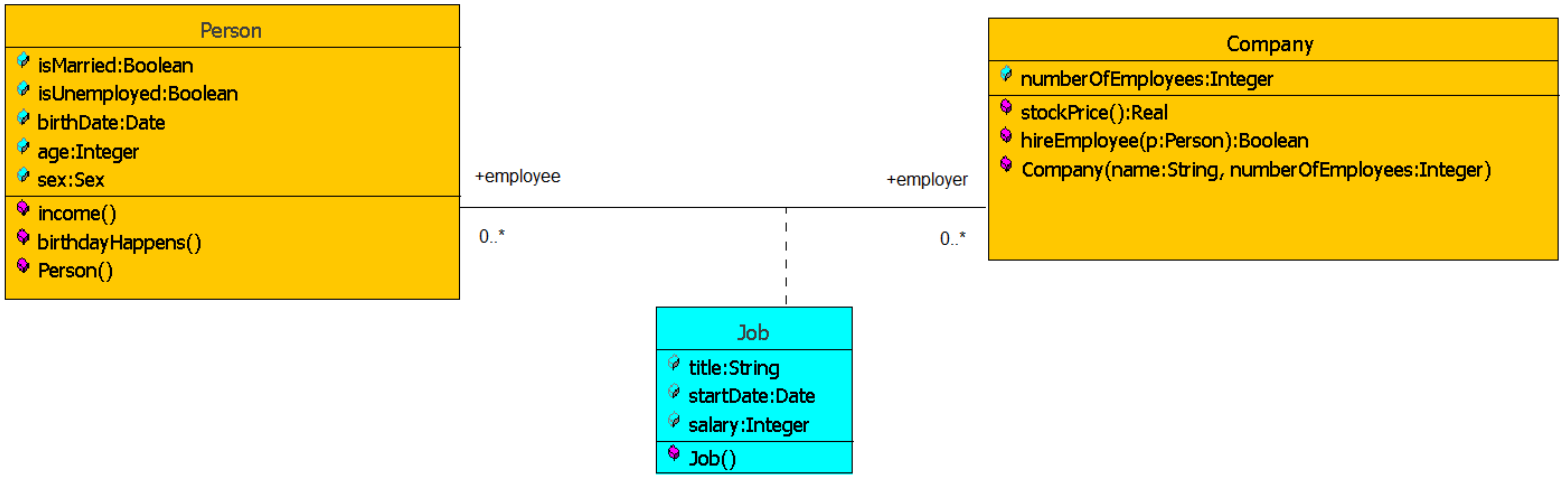
```java
public final void addCustomer(int accountNumber, Person arg) {
        if (arg != null) {
                ArrayList key = new ArrayList();
                key.add(Integer.toInteger(accountNumber));
                if (customer == null) customer = new HashMap();
                Person temp = (Person)customer.put(key, arg); //the previous value, if any
                if (temp != arg) {
                        arg.setBank(this);
                        if (temp != null) {
                                temp.setBank(null);
                        }
                }
        }
}
```

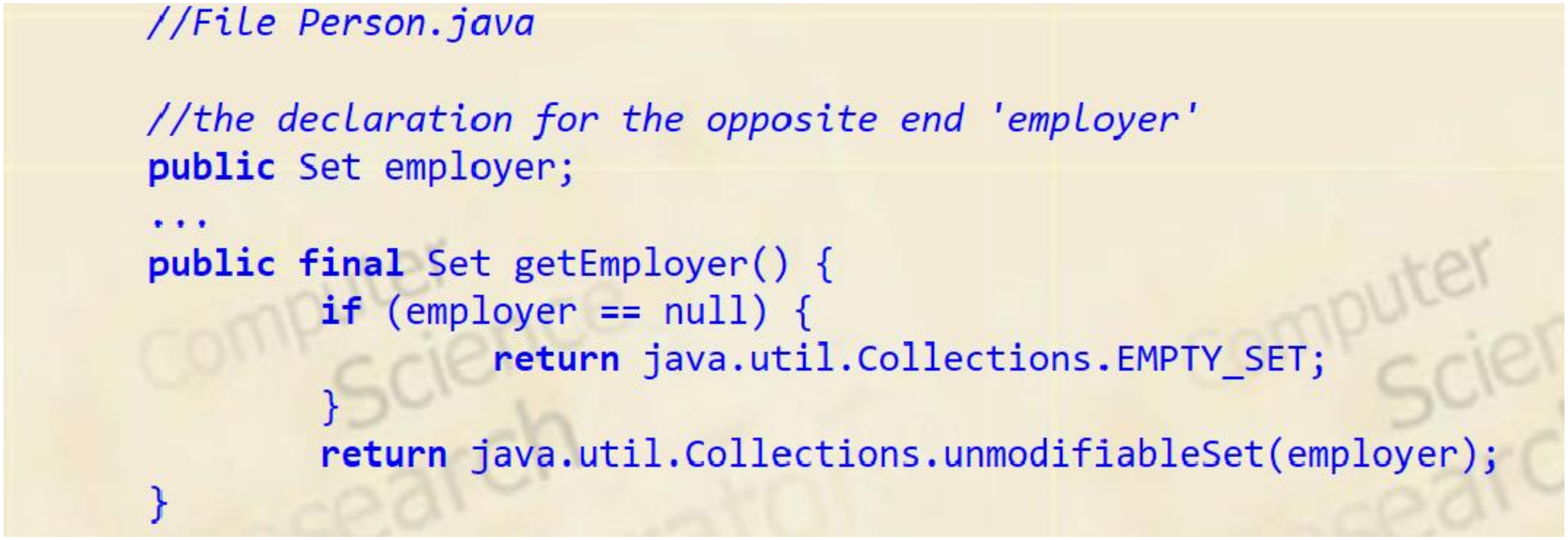
Bidirectional aualified association



```
public final void removeCustomer(int accountNumber) {
        if (customer != null) {
                ArrayList key = new ArrayList();
                key.add(Integer.toInteger(accountNumber));
                Person temp = (Person)customer.remove(key);
                if (temp != null) {
                        temp.setBank(null);
                }
        }
}

public final void removeCustomer(Person arg) {
        if (customer != null || arg != null) {
                if (customer.values().remove(arg)) {
                        arg.setBank(null);
                }
        }
}
```

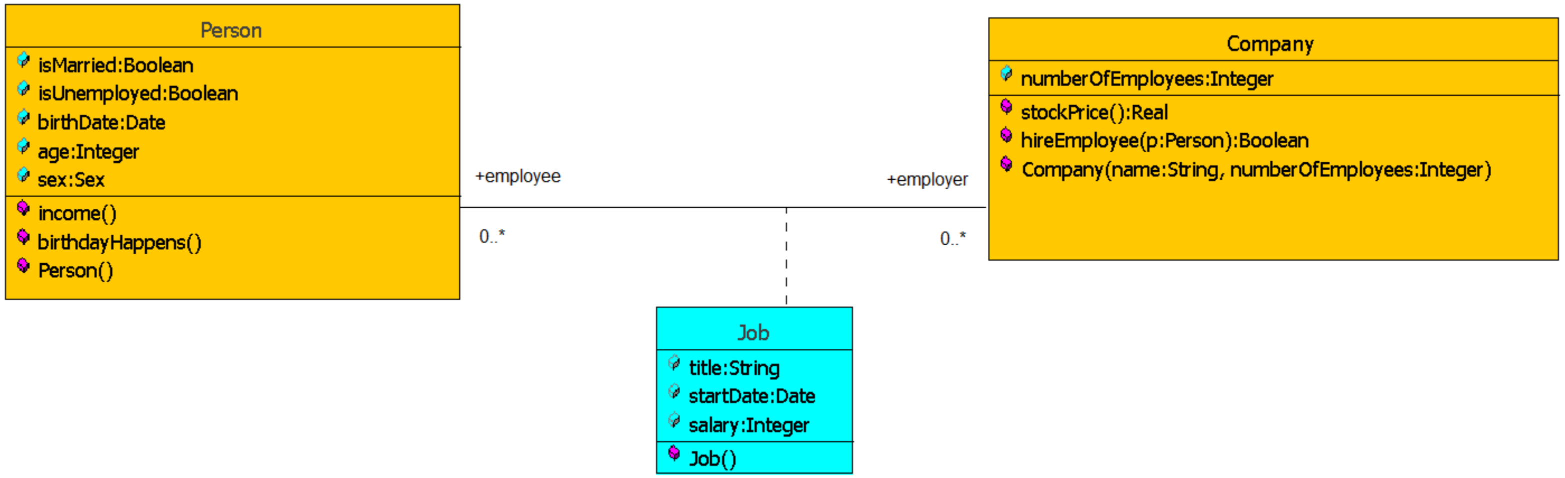
Association class

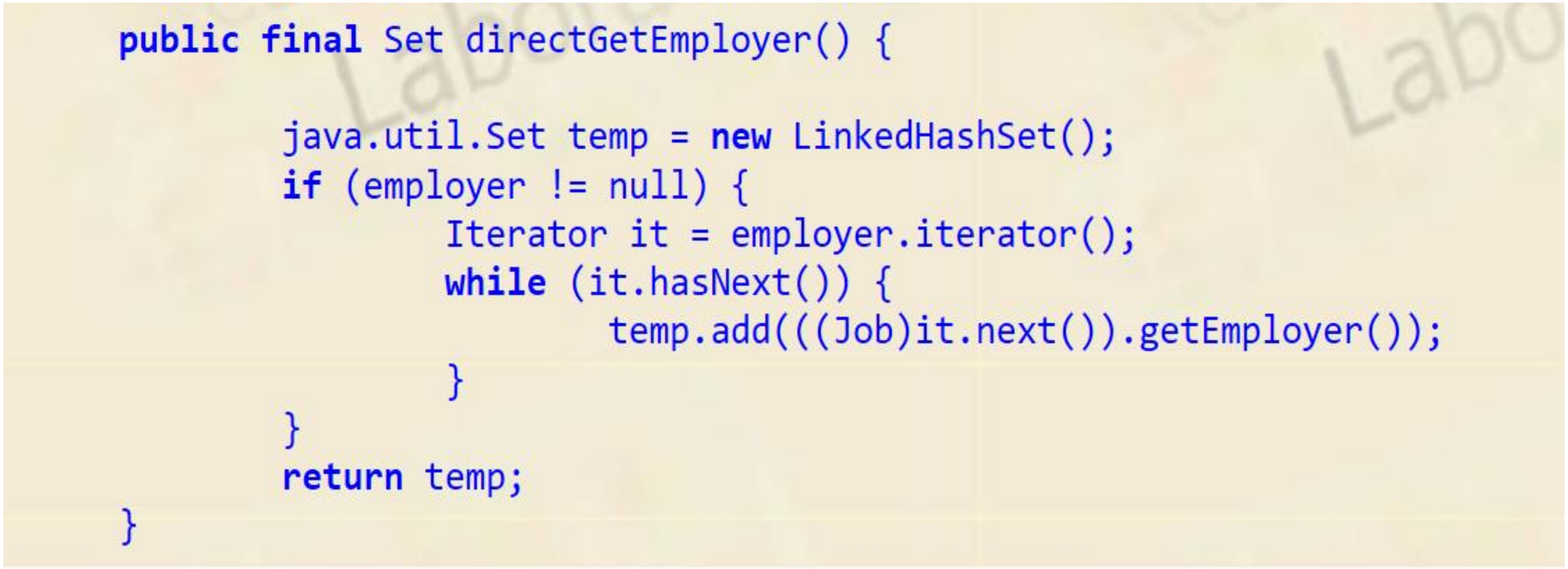




```
//File Person.java

//the declaration for the opposite end 'employer'
public Set employer;
...
public final Set getEmployer() {
        if (employer == null) {
                return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(employer);
}
```

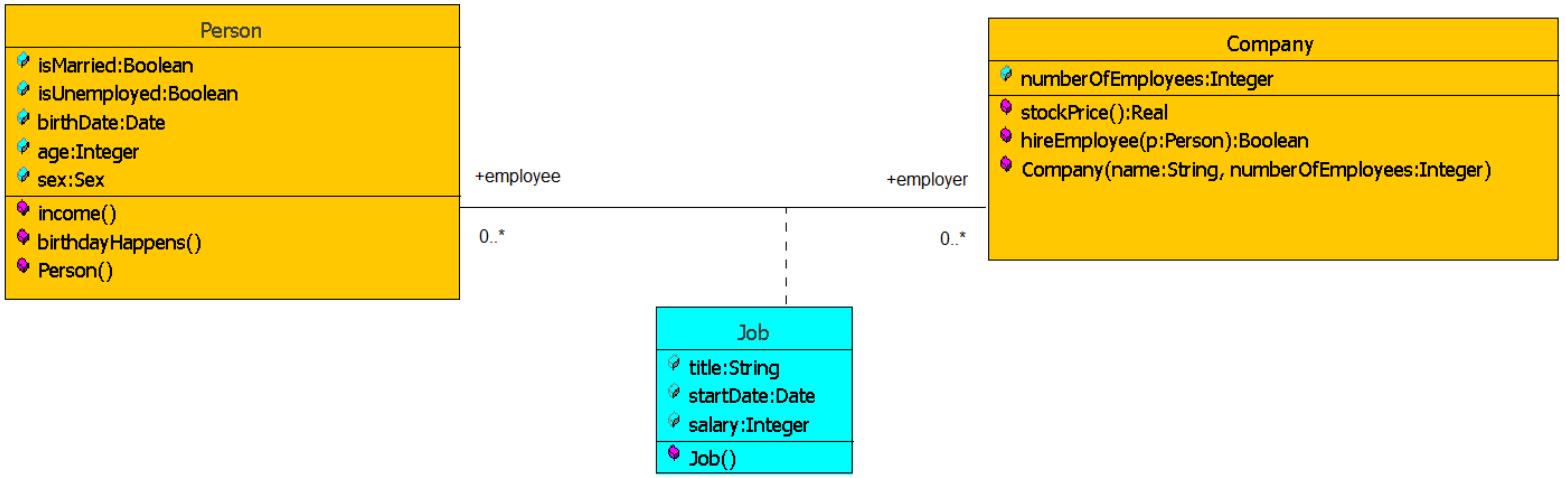
Association class

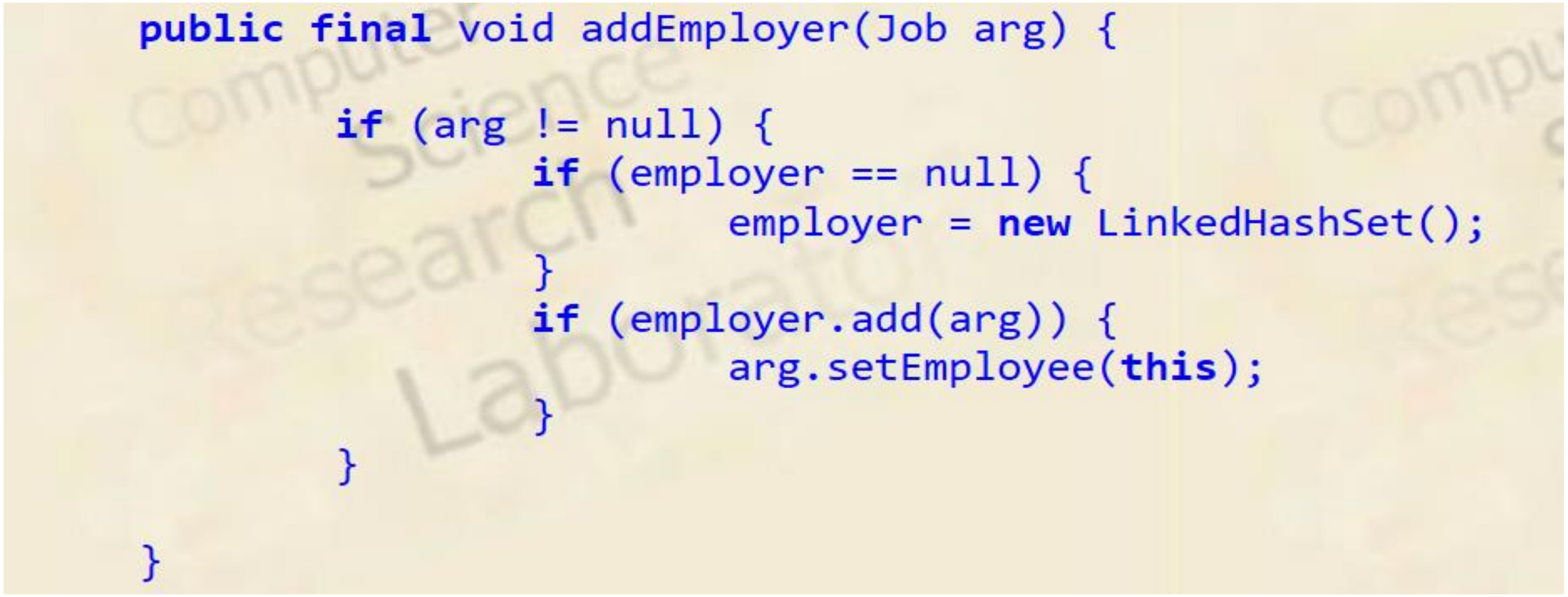




```
public final Set directGetEmployer() {

        java.util.Set temp = new LinkedHashSet();
        if (employer != null) {
                Iterator it = employer.iterator();
                while (it.hasNext()) {
                        temp.add(((Job)it.next()).getEmployer());
                }
        }
        return temp;
}
```

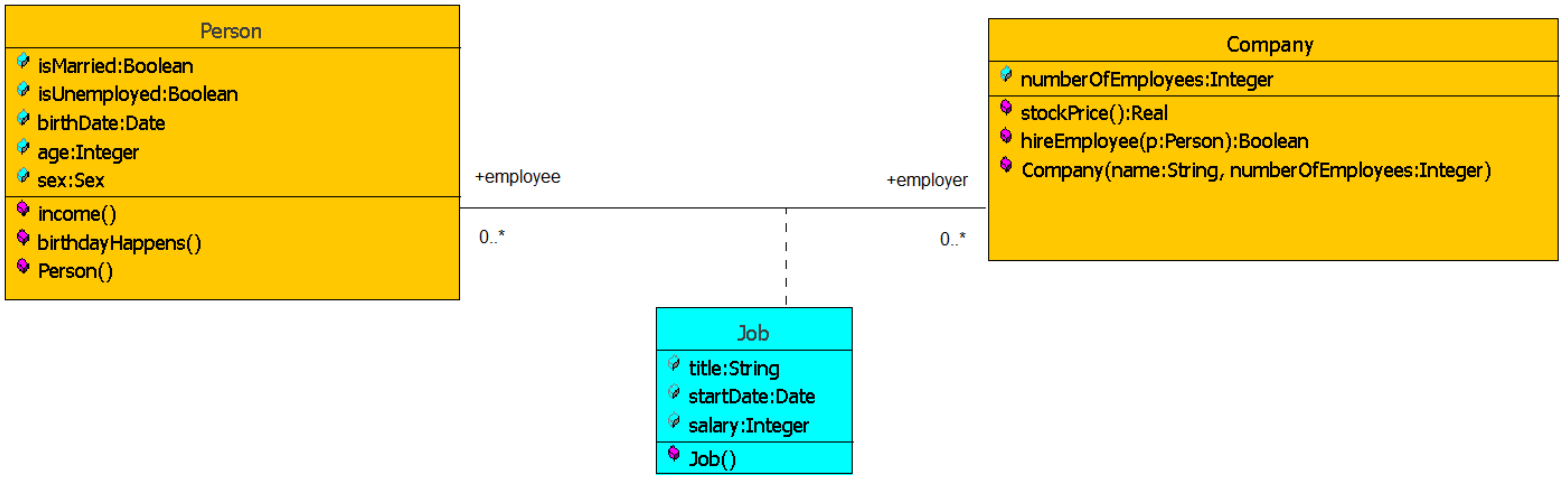
Association class

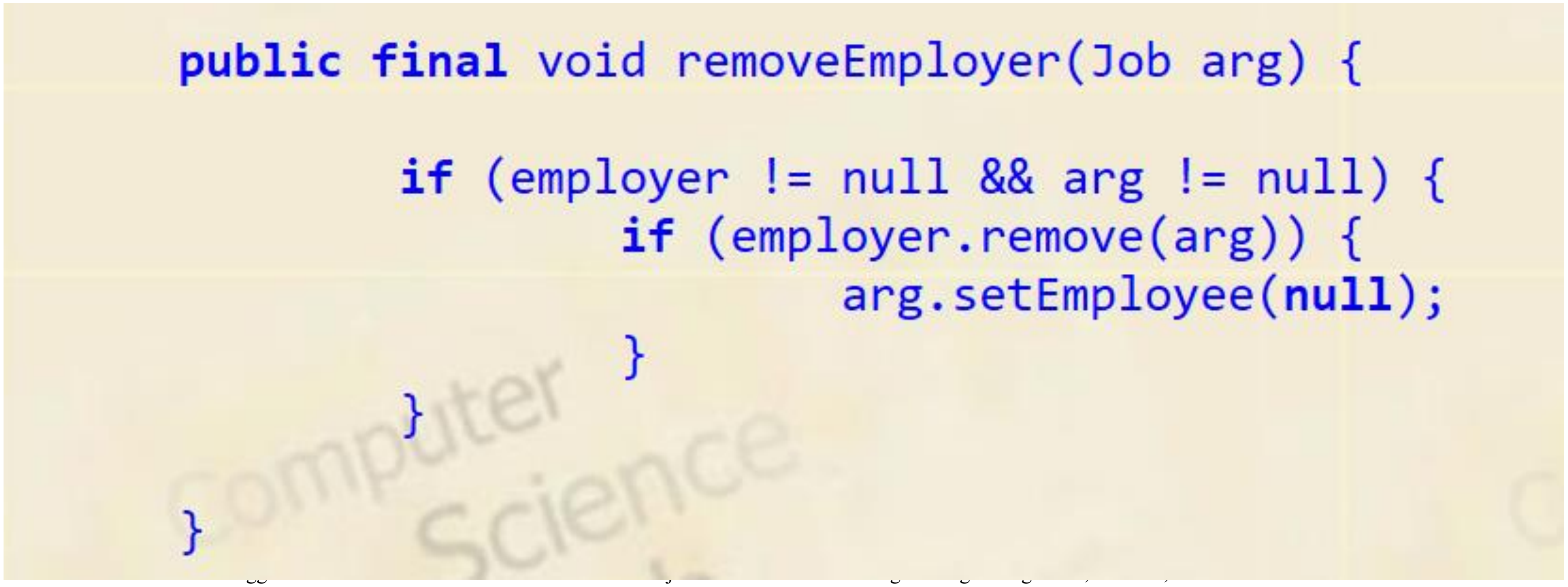




```
public final void addEmployer(Job arg) {

        if (arg != null) {
                if (employer == null) {
                        employer = new LinkedHashSet();
                }
                if (employer.add(arg)) {
                        arg.setEmployee(this);
                }
        }

}
```

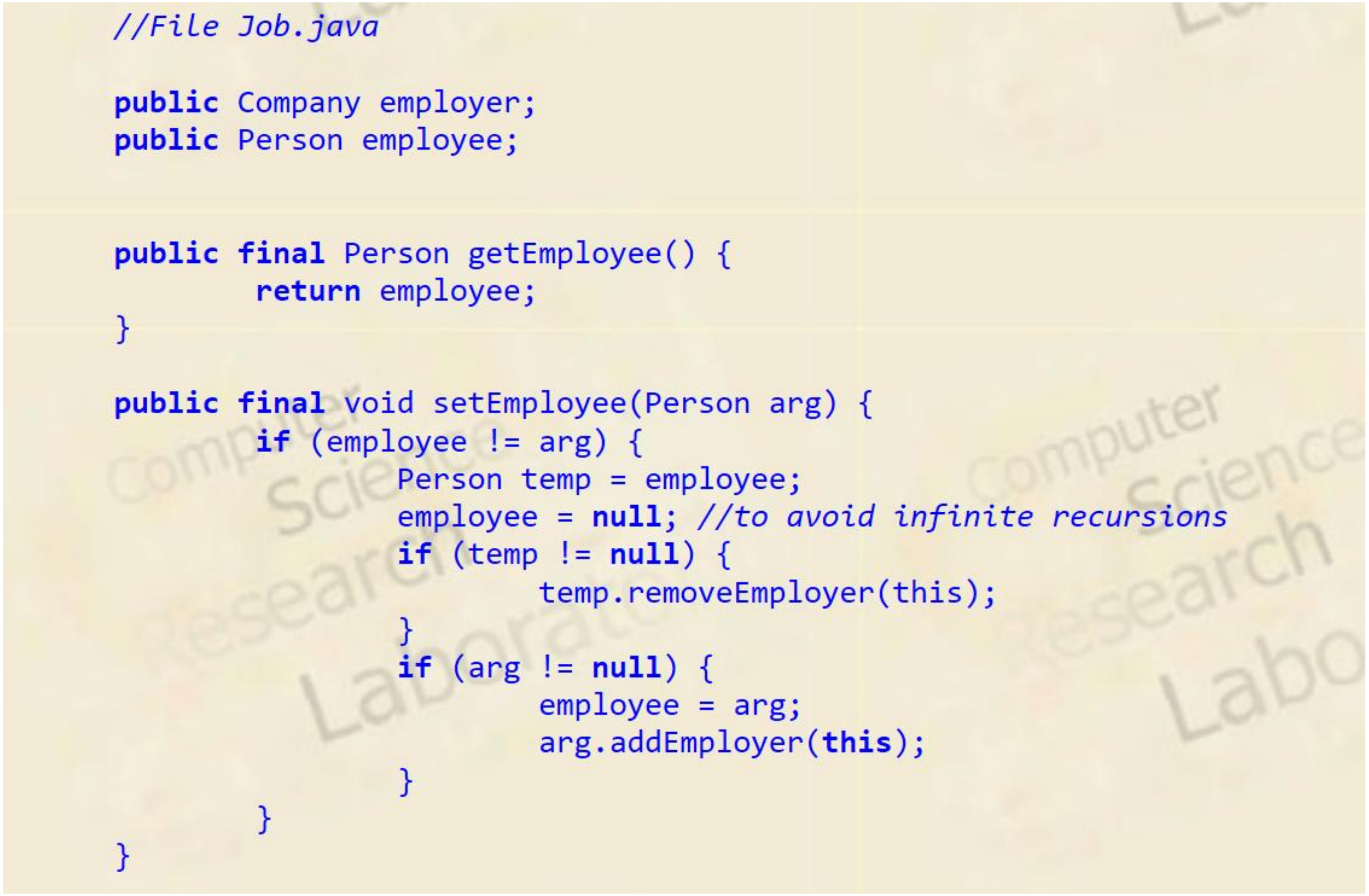
Association class





```
public final void removeEmployer(Job arg) {

        if (employer != null && arg != null) {
                if (employer.remove(arg)) {
                        arg.setEmployee(null);
                }
        }
}
```

Association class



```
//File Job.java

public Company employer;
public Person employee;


public final Person getEmployee() {
        return employee;
}

public final void setEmployee(Person arg) {
        if (employee != arg) {
                Person temp = employee;
                employee = null; //to avoid infinite recursions
                if (temp != null) {
                        temp.removeEmployer(this);
                }
                if (arg != null) {
                        employee = arg;
                        arg.addEmployer(this);
                }
        }
}
```

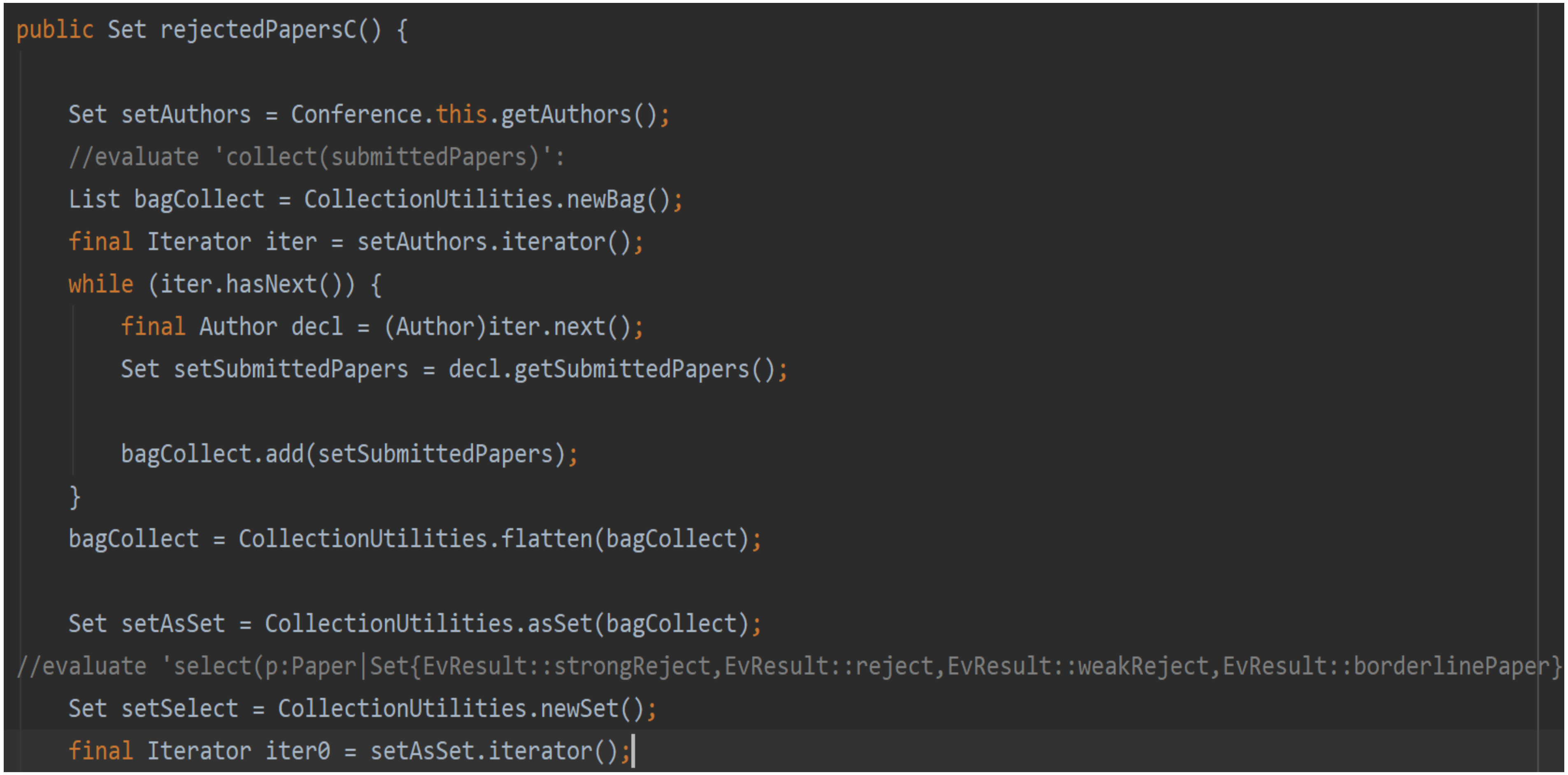
Examples of Model Transformations and Forward Engineering

- Model Transformations
  - Goal: Optimizing the object design model
    - ✓ Collapsing objects
    - ✓ Delaying expensive computations
- Forward Engineering
  - Goal: Implementing the object design model in a programming language
  - ✓ Mapping inheritance
  - ✓ Mapping associations
  - ➡ Transforming observers into code
  - Mapping contracts to exceptions
  - Mapping object models to tables

✓Transforming observers into code

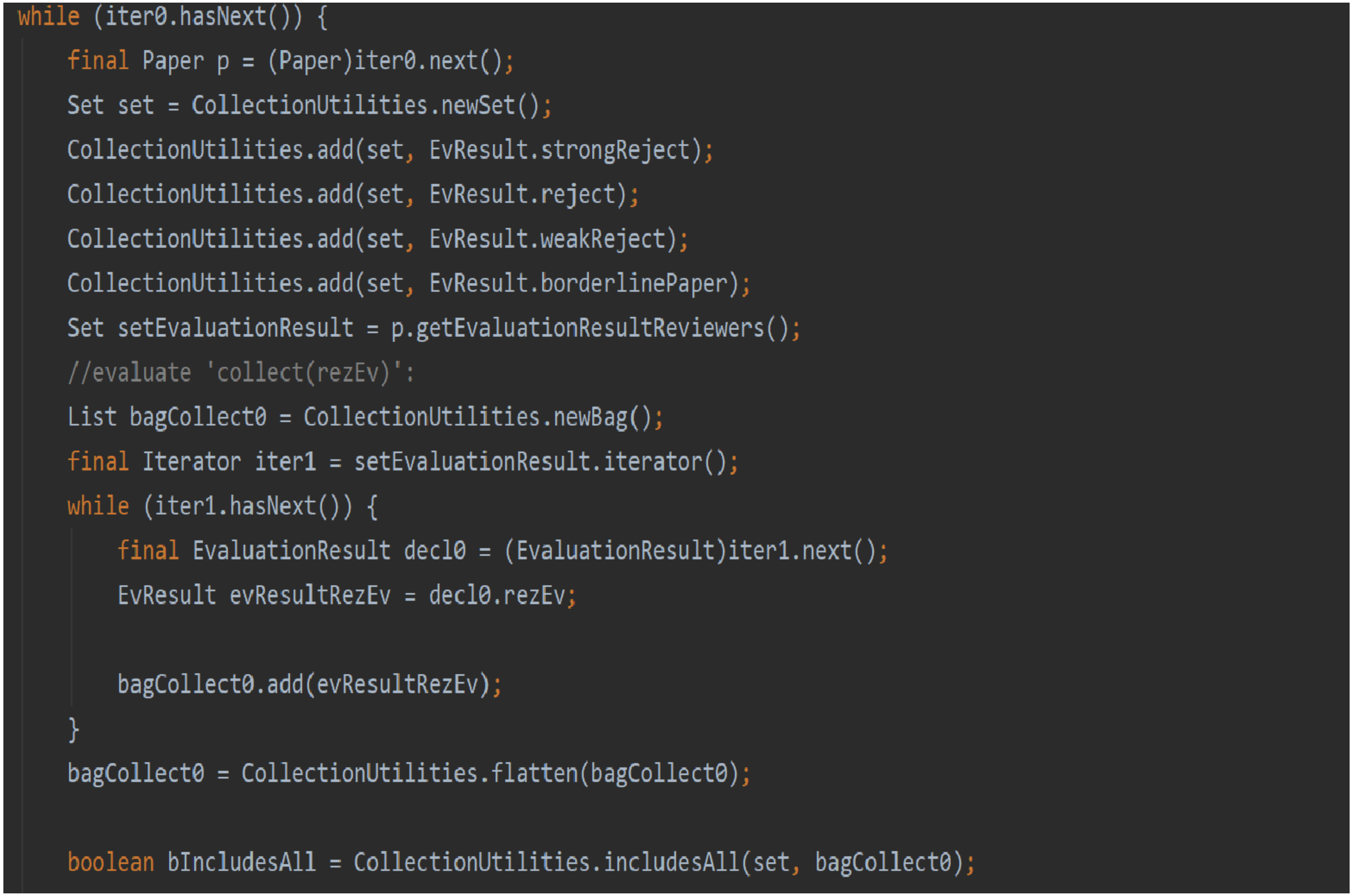
```
context Conference
  def allEvalResBorderline:
    let allEvalResBorderline:Set(Paper)=self.authors.submittedPapers
    ->asSet->select(p:Paper | p.evaluationResult.rezEv->forAll(rE |
    rE=EvResult::borderlinePaper

    let rejectedPapersC: Set(Paper) = self.authors.submittedPapers->
     asSet->select(p:Paper | Set{EvResult::strongReject,
     EvResult::reject, EvResult::weakReject,EvResult::borderlinePaper}
     ->includesAll(p.evaluationResult.rezEv))-allEvalResBorderline
```

✓ Transforming observers into code



```
public Set rejectedPapersC() {

    Set setAuthors = Conference.this.getAuthors();
    //evaluate 'collect(submittedPapers)':
    List bagCollect = CollectionUtilities.newBag();
    final Iterator iter = setAuthors.iterator();
    while (iter.hasNext()) {
        final Author decl = (Author)iter.next();
        Set setSubmittedPapers = decl.getSubmittedPapers();

        bagCollect.add(setSubmittedPapers);
    }
    bagCollect = CollectionUtilities.flatten(bagCollect);

    Set setAsSet = CollectionUtilities.asSet(bagCollect);
//evaluate 'select(p:Paper|Set{EvResult::strongReject,EvResult::reject,EvResult::weakReject,EvResult::borderlinePaper}
    Set setSelect = CollectionUtilities.newSet();
    final Iterator iter0 = setAsSet.iterator();
```

✓Transforming observers into code _ cont

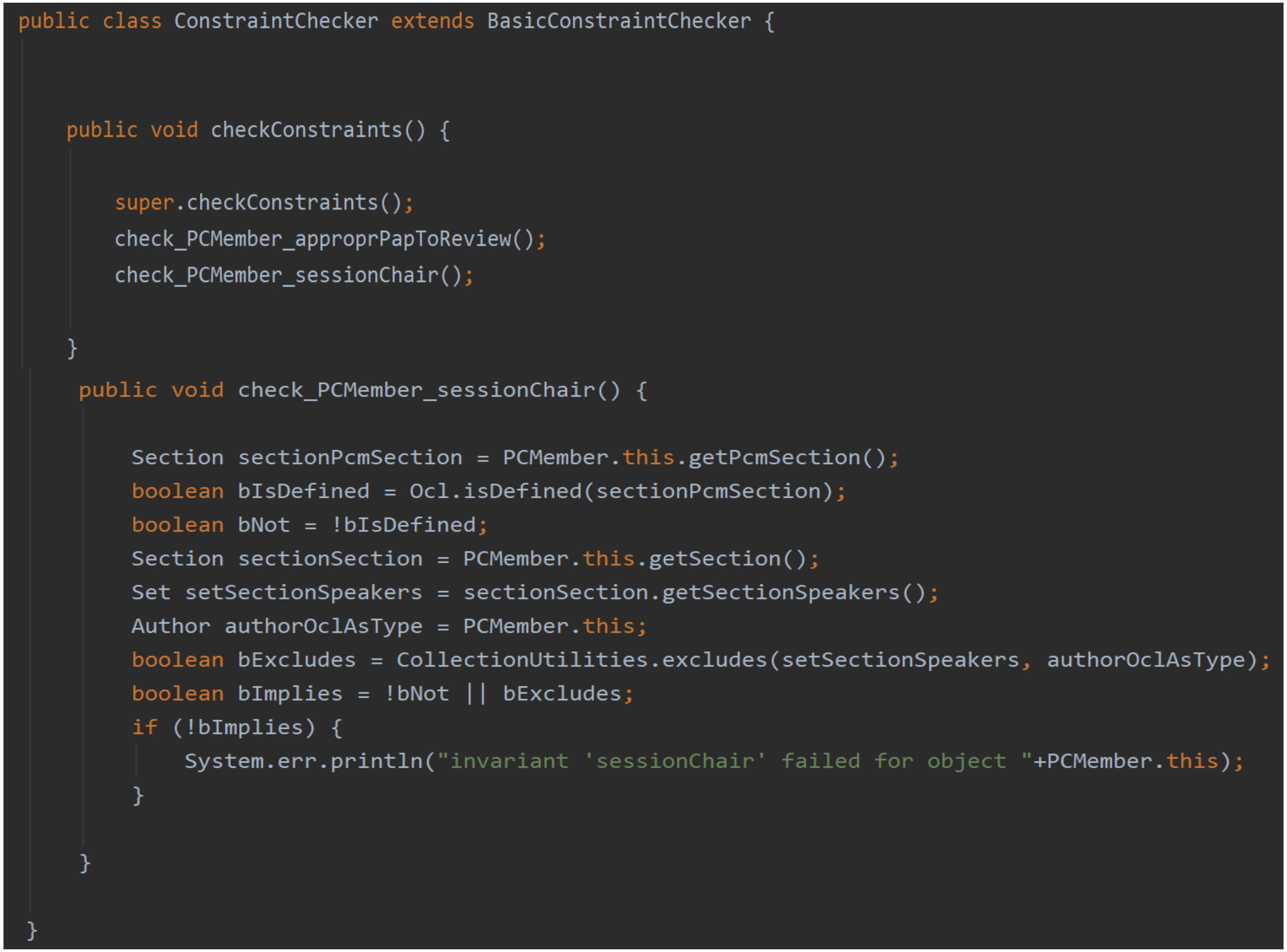


```
while (iter0.hasNext()) {
    final Paper p = (Paper)iter0.next();
    Set set = CollectionUtilities.newSet();
    CollectionUtilities.add(set, EvResult.strongReject);
    CollectionUtilities.add(set, EvResult.reject);
    CollectionUtilities.add(set, EvResult.weakReject);
    CollectionUtilities.add(set, EvResult.borderlinePaper);
    Set setEvaluationResult = p.getEvaluationResultReviewers();
    //evaluate 'collect(rezEv)':
    List bagCollect0 = CollectionUtilities.newBag();
    final Iterator iter1 = setEvaluationResult.iterator();
    while (iter1.hasNext()) {
        final EvaluationResult decl0 = (EvaluationResult)iter1.next();
        EvResult evResultRezEv = decl0.rezEv;

        bagCollect0.add(evResultRezEv);
    }
    bagCollect0 = CollectionUtilities.flatten(bagCollect0);

    boolean bIncludesAll = CollectionUtilities.includesAll(set, bagCollect0);
```

Implementing Contract Violations

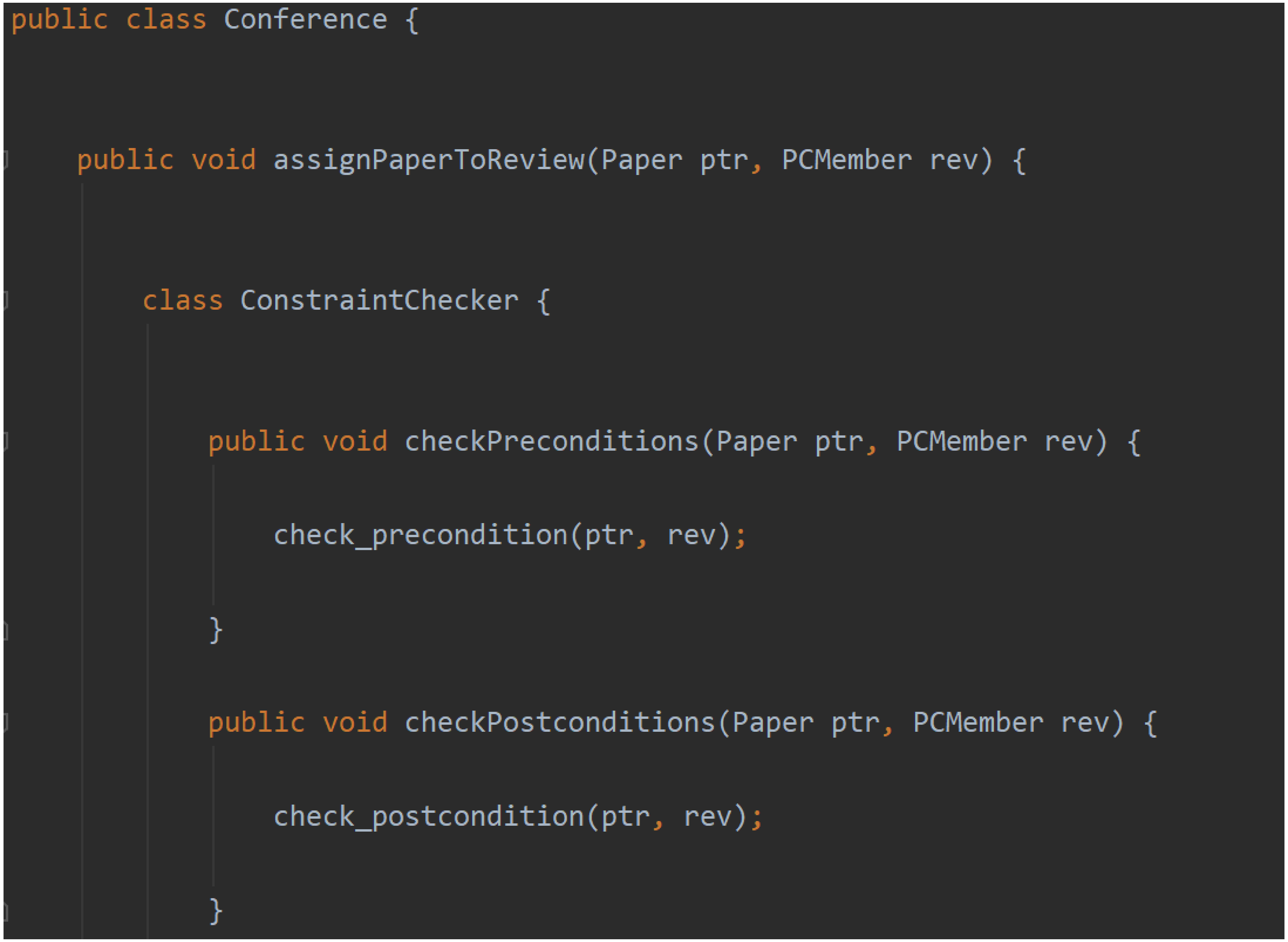
- Many object-oriented languages do not have built-in support for contracts
- However, if they support exceptions, we can use their exception mechanisms for signaling and handling contract violations
- In Java we use the try-throw-catch mechanism
- Example:
  - Let us assume the acceptPlayer() operation of TournamentControl is invoked with a player who is already part of the Tournament
    - UML model (see slide 61)
  - In this case acceptPlayer() in TournamentControl should throw an exception of type KnownPlayer
    - Java Source code (see slide 62).

Implementing Contract Violations - invariants



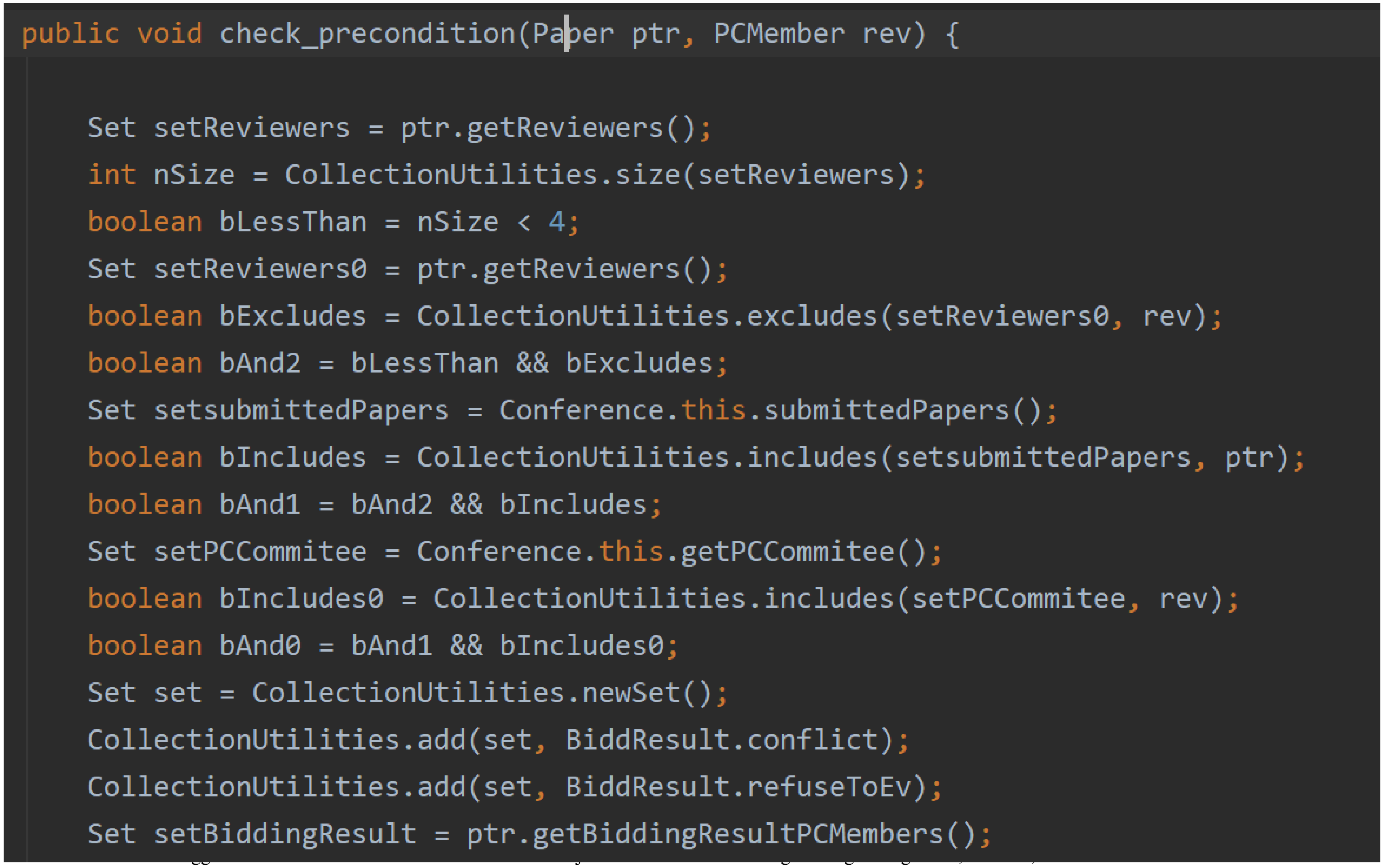
```java
public class ConstraintChecker extends BasicConstraintChecker {


    public void checkConstraints() {

        super.checkConstraints();
        check_PCMember_approprPapToReview();
        check_PCMember_sessionChair();

    }
     public void check_PCMember_sessionChair() {

         Section sectionPcmSection = PCMember.this.getPcmSection();
         boolean bIsDefined = Ocl.isDefined(sectionPcmSection);
         boolean bNot = !bIsDefined;
         Section sectionSection = PCMember.this.getSection();
         Set setSectionSpeakers = sectionSection.getSectionSpeakers();
         Author authorOclAsType = PCMember.this;
         boolean bExcludes = CollectionUtilities.excludes(setSectionSpeakers, authorOclAsType);
         boolean bImplies = !bNot || bExcludes;
         if (!bImplies) {
             System.err.println("invariant 'sessionChair' failed for object "+PCMember.this);
         }

     }

 }
```

Implementing Contract Violations – pre&post



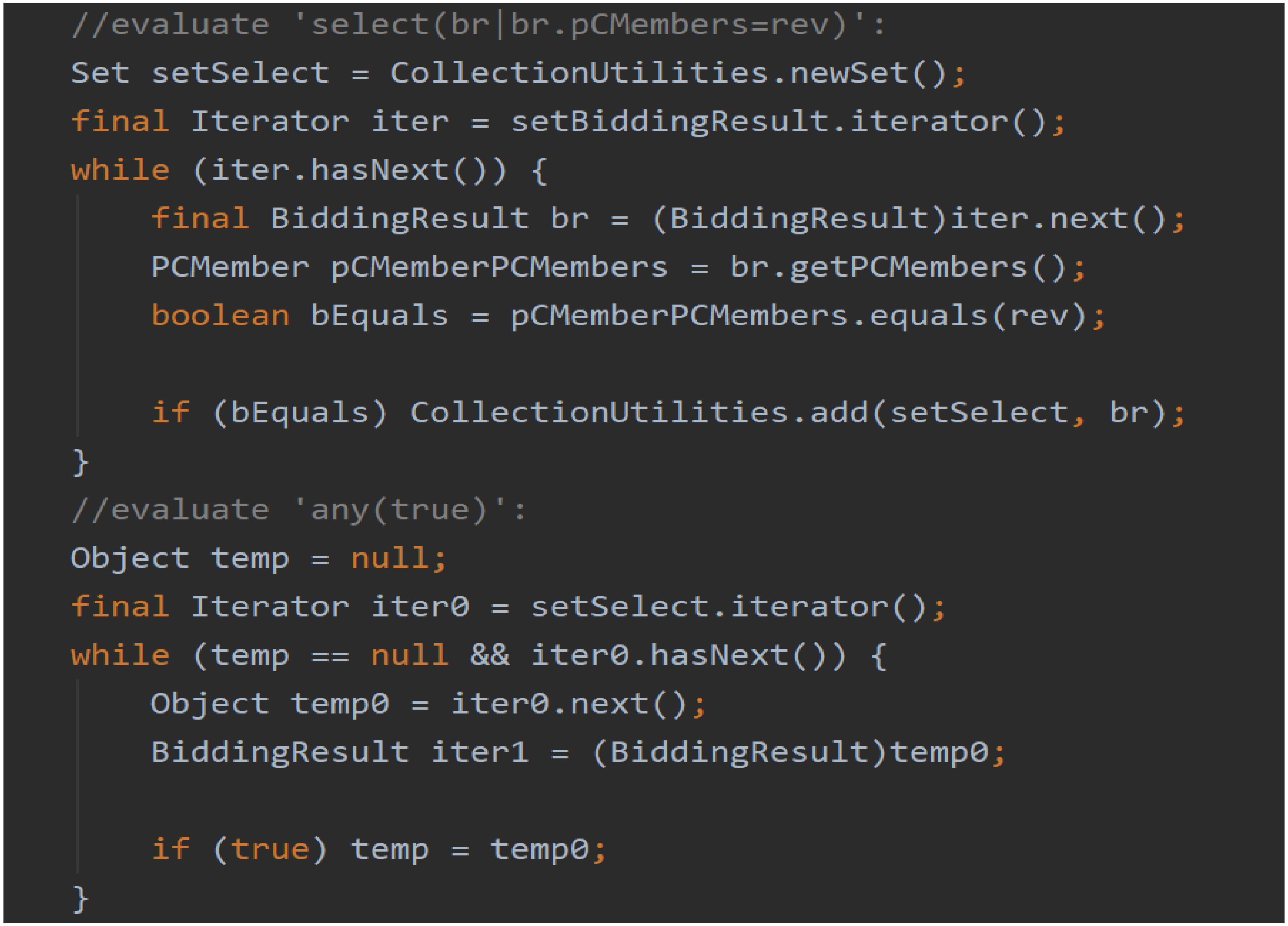
```
public class Conference {


    public void assignPaperToReview(Paper ptr, PCMember rev) {


        class ConstraintChecker {


            public void checkPreconditions(Paper ptr, PCMember rev) {

                check_precondition(ptr, rev);

            }

            public void checkPostconditions(Paper ptr, PCMember rev) {

                check_postcondition(ptr, rev);

            }
```

Implementing Contract Violations – pre



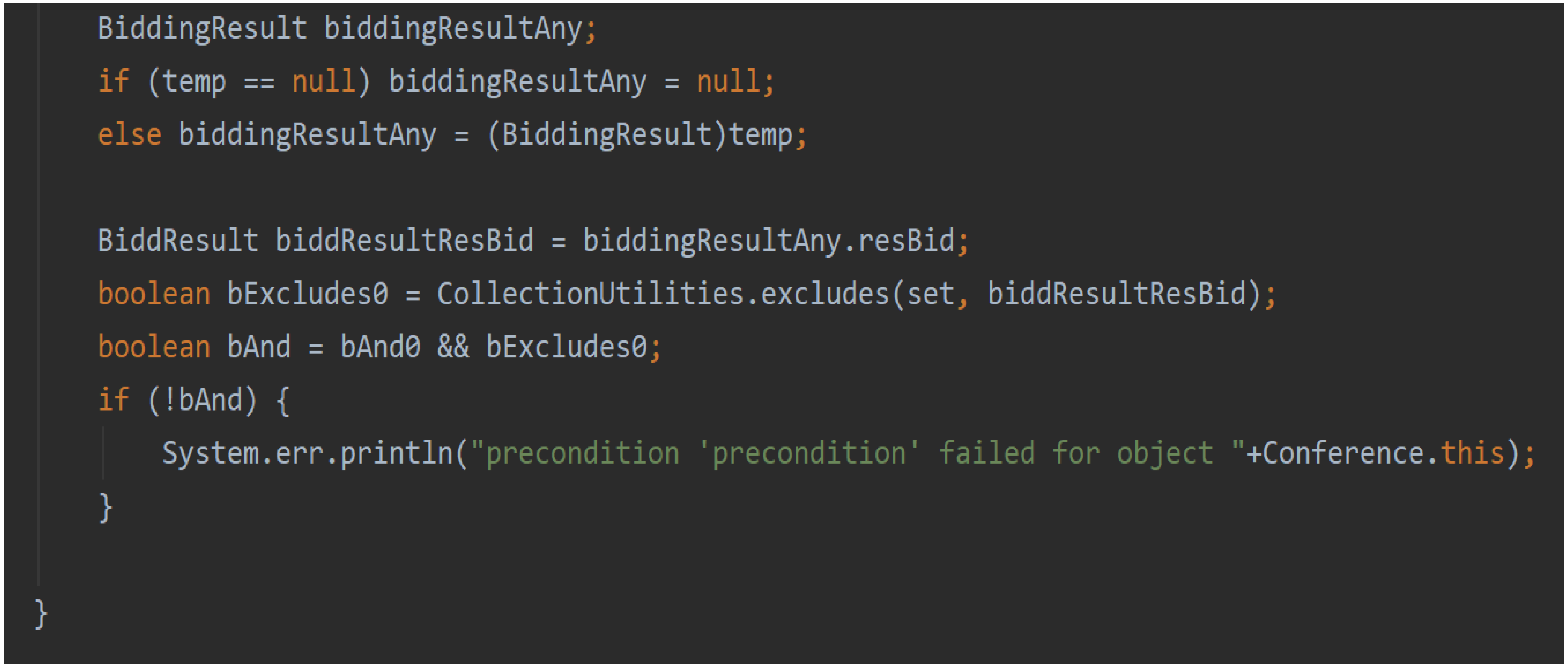
```
public void check_precondition(Paper ptr, PCMember rev) {

    Set setReviewers = ptr.getReviewers();
    int nSize = CollectionUtilities.size(setReviewers);
    boolean bLessThan = nSize < 4;
    Set setReviewers0 = ptr.getReviewers();
    boolean bExcludes = CollectionUtilities.excludes(setReviewers0, rev);
    boolean bAnd2 = bLessThan && bExcludes;
    Set setsubmittedPapers = Conference.this.submittedPapers();
    boolean bIncludes = CollectionUtilities.includes(setsubmittedPapers, ptr);
    boolean bAnd1 = bAnd2 && bIncludes;
    Set setPCCommitee = Conference.this.getPCCommitee();
    boolean bIncludes0 = CollectionUtilities.includes(setPCCommitee, rev);
    boolean bAnd0 = bAnd1 && bIncludes0;
    Set set = CollectionUtilities.newSet();
    CollectionUtilities.add(set, BiddResult.conflict);
    CollectionUtilities.add(set, BiddResult.refuseToEv);
    Set setBiddingResult = ptr.getBiddingResultPCMembers();
```

Implementing Contract Violations – pre_2



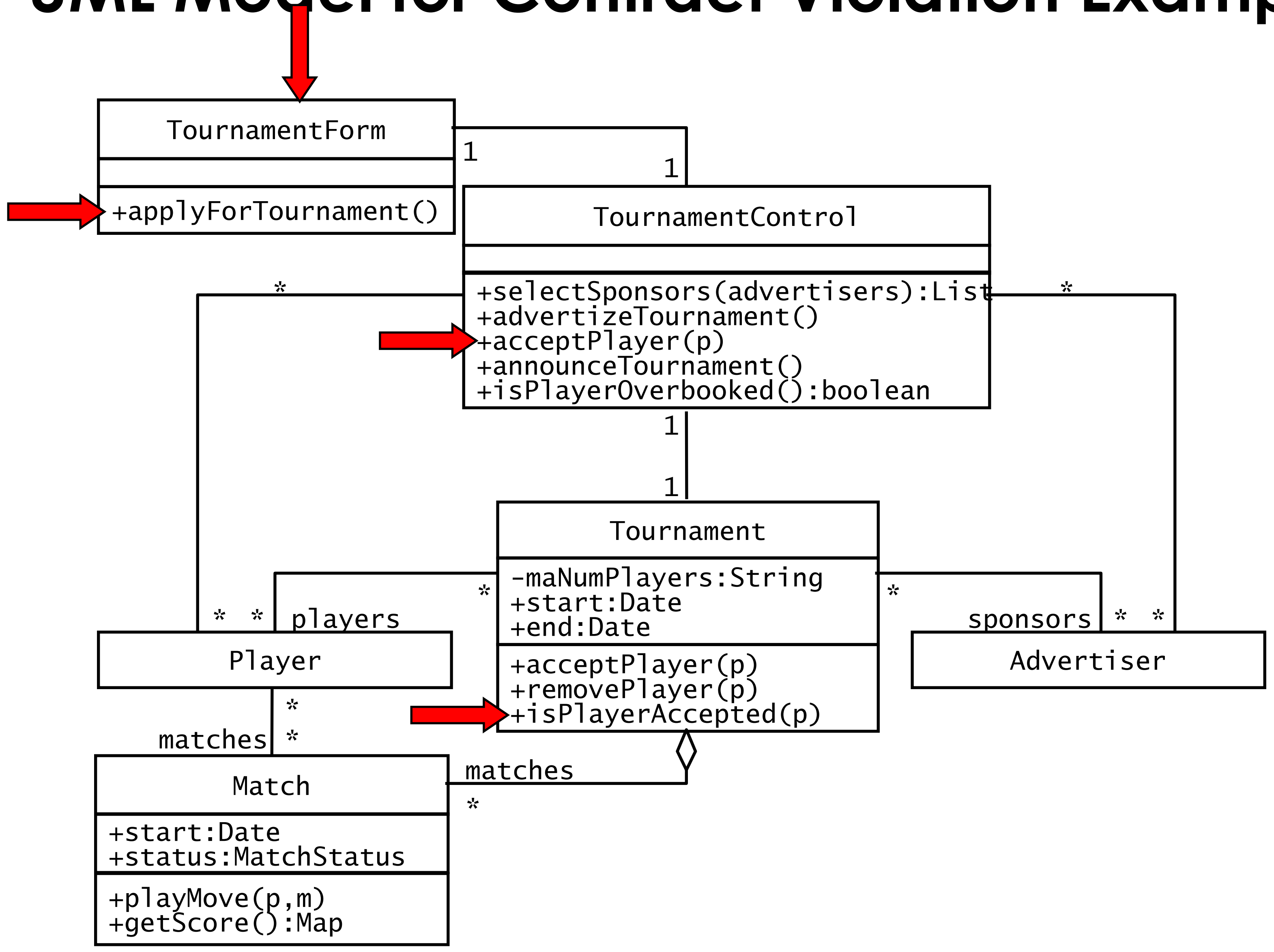
```java
//evaluate 'select(br|br.pCMembers=rev)':
Set setSelect = CollectionUtilities.newSet();
final Iterator iter = setBiddingResult.iterator();
while (iter.hasNext()) {
    final BiddingResult br = (BiddingResult)iter.next();
    PCMember pCMemberPCMembers = br.getPCMembers();
    boolean bEquals = pCMemberPCMembers.equals(rev);

    if (bEquals) CollectionUtilities.add(setSelect, br);
}
//evaluate 'any(true)':
Object temp = null;
final Iterator iter0 = setSelect.iterator();
while (temp == null && iter0.hasNext()) {
    Object temp0 = iter0.next();
    BiddingResult iter1 = (BiddingResult)temp0;

    if (true) temp = temp0;
}
```

Implementing Contract Violations – pre_3

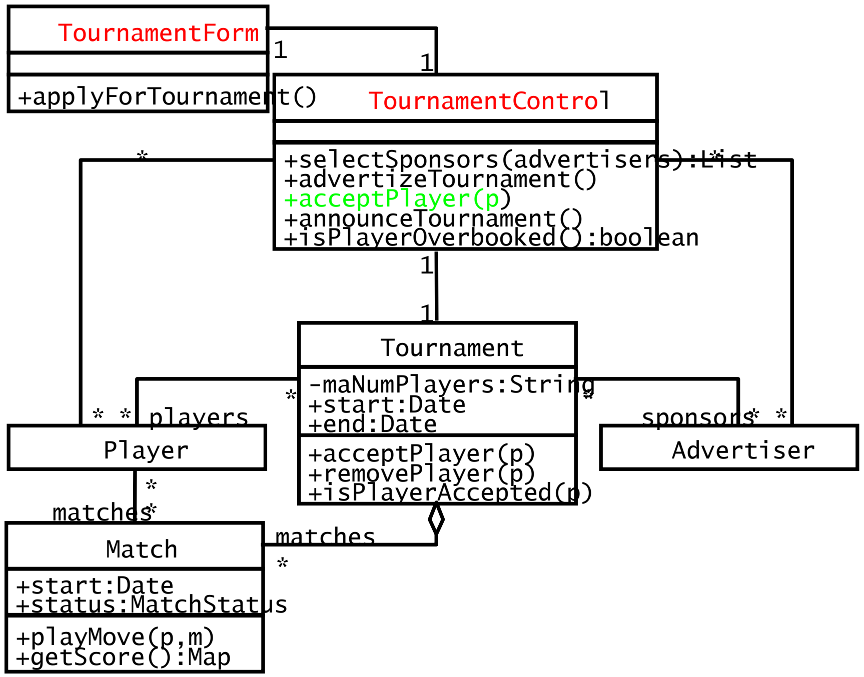


```
    BiddingResult biddingResultAny;
    if (temp == null) biddingResultAny = null;
    else biddingResultAny = (BiddingResult)temp;

    BiddResult biddResultResBid = biddingResultAny.resBid;
    boolean bExcludes0 = CollectionUtilities.excludes(set, biddResultResBid);
    boolean bAnd = bAnd0 && bExcludes0;
    if (!bAnd) {
        System.err.println("precondition 'precondition' failed for object "+Conference.this);
    }

}
```

UML Model for Contract Violation Example



Implementation in Java



```
public class TournamentForm {
  private TournamentControl control;
  private ArrayList players;
  public void processPlayerApplications() {
   for (Iteration i = players.iterator(); i.hasNext();) {
        try {
             control.acceptPlayer((Player)i.next());
         }
        catch (KnownPlayerException e) {
          // If exception was caught, log it to console
          ErrorConsole.log(e.getMessage());
          }
         }
       }
}
```

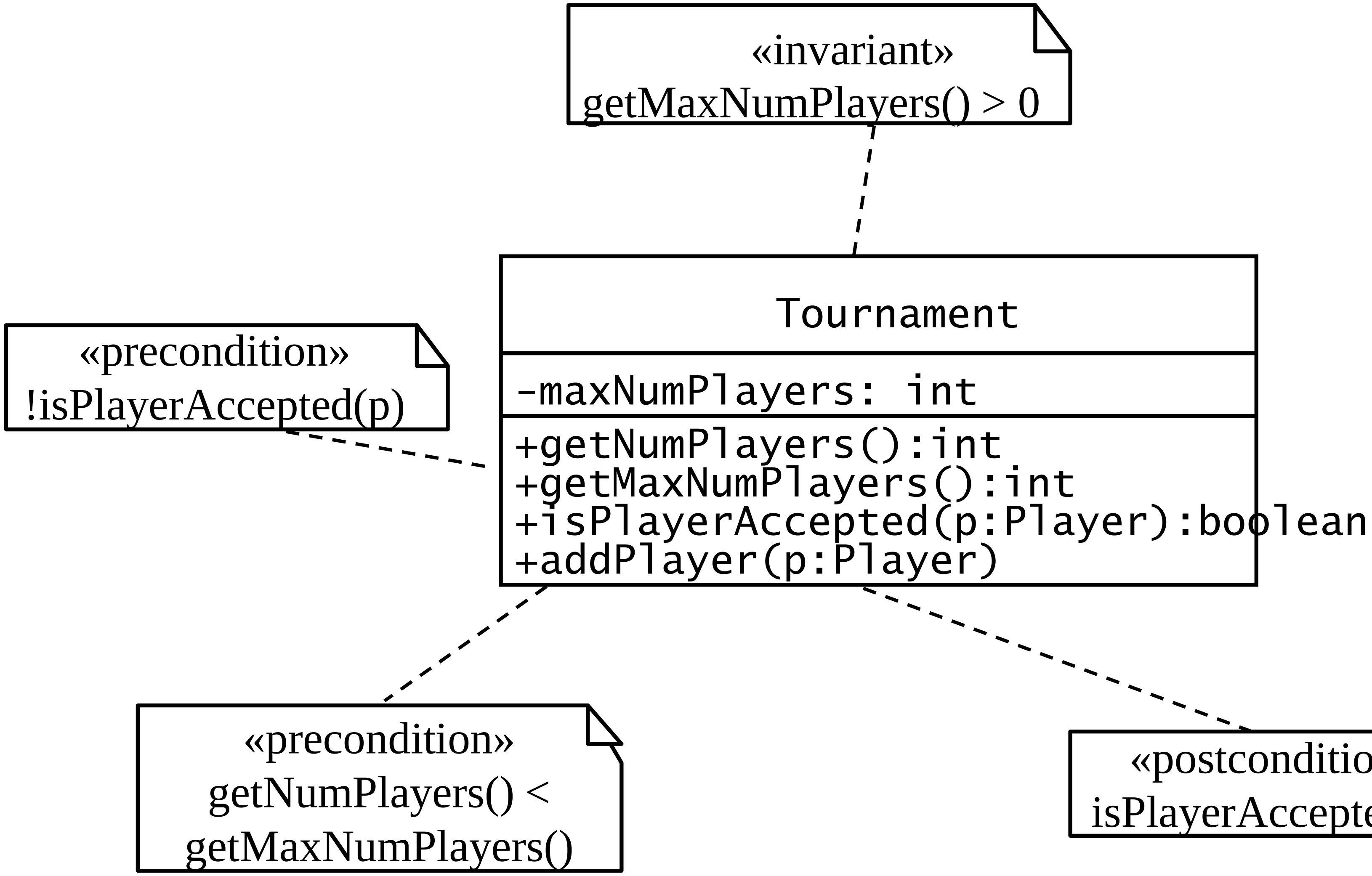
The try-throw-catch Mechanism in Java

```
public class TournamentControl {
  private Tournament tournament;
  public void addPlayer(Player p) throws KnownPlayerException
  {
        if (tournament.isPlayerAccepted(p)) {
             throw new KnownPlayerException(p);
        }
        //... Normal addPlayer behavior
  }
}
public class TournamentForm {
  private TournamentControl control;
  private ArrayList players;
  public void processPlayerApplications() {
   for (Iteration i = players.iterator(); i.hasNext();) {
        try {
             control.acceptPlayer((Player)i.next());
         }
        catch (KnownPlayerException e) {
          // If exception was caught, log it to console
          ErrorConsole.log(e.getMessage());
          }
         }
        }
}
```

Implementing a Contract

- **Check each precondition**:
  - Before the beginning of the method with a test to check the precondition for that method
    - Raise an exception if the precondition evaluates to false
- **Check each postcondition:**
  - At the end of the method write a test to check the postcondition
    - Raise an exception if the postcondition evaluates to false. If more than one postcondition is not satisfied, raise an exception only for the first violation.
- **Check each invariant:**
  - Check invariants at the same time when checking preconditions and when checking postconditions
- **Deal with inheritance:**
  - Add the checking code for preconditions and postconditions also into methods that can be called from the class.

A complete implementation of the Tournament.addPlayer() contract



Heuristics: Mapping Contracts to Exceptions

- Executing checking code slows down your program
  - If it is too slow, omit the checking code for private and protected methods
  - If it is still too slow, focus on components with the longest life
    - Omit checking code for postconditions and invariants for all other components.

Heuristics for Transformations

- For any given transformation always use the same tool
- Keep the contracts in the source code, not in the object design model
- Use the same names for the same objects
- Have a style guide for transformations (Martin Fowler)

Object Design Areas

1. Service specification
   - Describes precisely each class interface
2. Component selection
   - Identify off-the-shelf components and additional solution objects
3. Object model restructuring
   - Transforms the object design model to improve its understandability and extensibility
4. Object model optimization
   - Transforms the object design model to address performance criteria such as response time or memory utilization.

Design Optimizations

- Design optimizations are an important part of the object design phase:
  - The requirements analysis model is semantically correct but often too inefficient if directly implemented.
- Optimization activities during object design:
  1. Add redundant associations to minimize access cost
  2. Rearrange computations for greater efficiency
  3. Store derived attributes to save computation time
- As an object designer you must strike a balance between efficiency and clarity.
  - Optimizations will make your models more obscure

Design Optimization Activities

1. Add redundant associations:
   - What are the most frequent operations? (Sensor data lookup?)
   - How often is the operation called? (30 times a month, every 50 milliseconds)
2. Rearrange execution order
   - Eliminate dead paths as early as possible (Use knowledge of distributions, frequency of path traversals)
   - Narrow search as soon as possible
   - Check if execution order of loop should be reversed
3. Turn classes into attributes

Implement application domain classes

- To collapse or not collapse: Attribute or association?
- Object design choices:
  - Implement entity as embedded attribute
  - Implement entity as separate class with associations to other classes
- Associations are more flexible than attributes but often introduce unnecessary indirection
- Abbott's textual analysis rules.

To Collapse or not to Collapse?

- Collapse a class into an attribute if the only operations defined on the attributes are Set() and Get().

Design Optimizations (continued)

Store derived attributes

- Example: Define new classes to store information locally (database cache)

- Problem with derived attributes:
  - Derived attributes must be updated when base values change.
  - There are 3 ways to deal with the update problem:
    - Explicit code: Implementor determines affected derived attributes (push)
    - Periodic computation: Recompute derived attribute occasionally (pull)
    - Active value: An attribute can designate set of dependent values which are automatically updated when active value is changed (notification, data trigger)