

# Software Engineering: Specification, Implementation, Verification

## Chapter 2: Specification of Constraints

Suad Alagić

*Springer 2017*

- Class invariants
- Preconditions and postconditions
- Constraints over collections
- Selection of collection elements
- Type conformance
- Queries
- Operations on collections
- Constraints and inheritance

# Specification of behavior

- Messages
- Method signatures
- Specification of behavior
- Constraint language
- Declarative specifications
- Requirements for code

# Specification of constraints

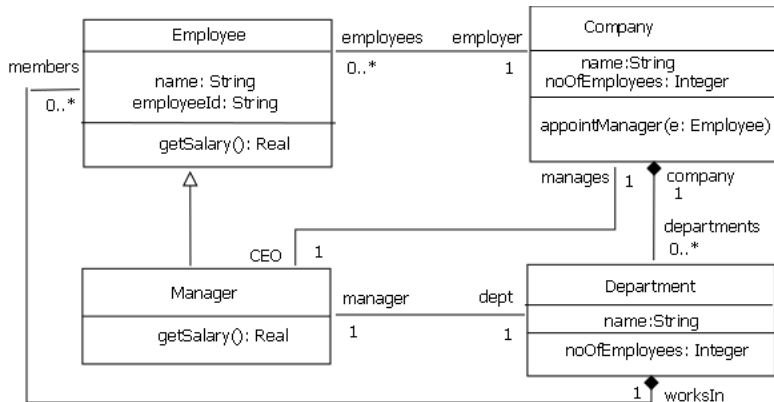


Figure: Company class diagram

- Properties of all objects of a class
- Context
- Expressions for constraints
- Reference to an object

# Class invariants

**context** Company **inv:**  
self.noOfEmployees  $\geq 0$

**context** Employee **inv**:  
self.salary()  $\geq$  10,000

**context** Employee **inv**:  
self.salary()  $\geq$  10,000 **and**  
self.salary()  $\leq$  100,000

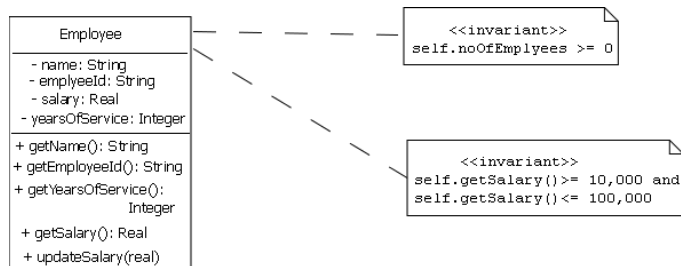


Figure: Invariant of class Employee



**context** Company **inv**:  
self.CEO.getSalary()  $\geq$  100,000

**context** Company **inv**:  
self.employees –  $>$  count() = self.noOfEmployees

**context** Company **inv**:  
self.employees –  $>$  notempty()

# Constraints for methods

- Context
- Preconditions
- Postconditions
- Reference to the previous state

# Pre- and post-conditions

```
context Employee:: hire(c: Company)
  pre not (c.employees— >includes(self))
  post c.employees— >includes(self)
  post c.noOfEmployees=c.noOfEmployees@pre +1
```

```
context Employee:: fire()
  pre self.employer.employees— >includes(self)
  post not(self.employer.employees— >includes(self))
  post self.employer.noOfEmployees=
    self.employer.noOfEmployees@pre - 1
```

# Preconditions and postconditions

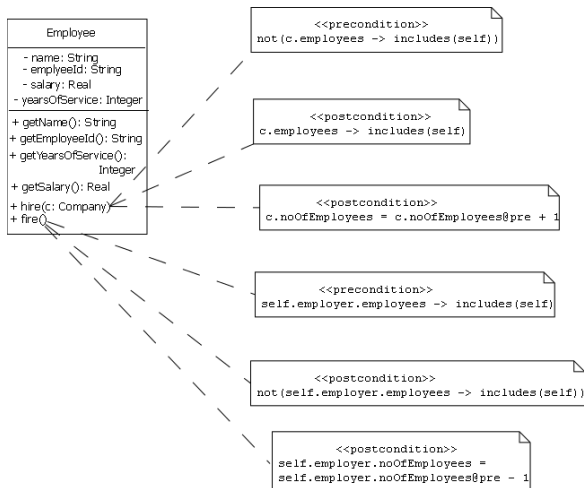


Figure: Pre and post conditions in UML diagrams

```
context Company:: appointManager(e: Employee)
  pre self.employees – >includes(e)
  post self.CEO = e
```

```
context Company :: selectManager(): Manager
  post result = self.CEO
```

# Constraints over collections

- Universal quantification
- Existential quantification
- Complex constraints
- Operators
- Constraints over all instances

# Constraints over collections

**context** Company **inv**:

self.departments – > **forAll**(d: Department | d.manager.getSalary()

**context** Company **inv**:

self.employees – > **forAll**(e1,e2: Employee |  
e1.employeeld = e2.employeeld **implies** e1=e2)

# Class assertions

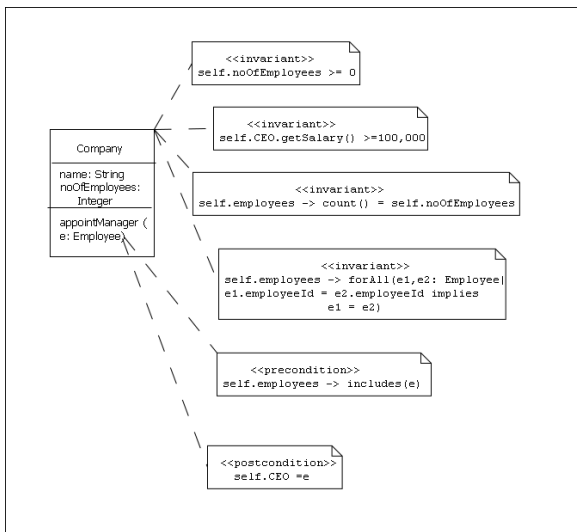


Figure: Assertions for the class Company



# Selection of elements

- Selection
- Selection and projection

# Constraints over collections

**context** Company **inv**:

self.employees – **>exists**(e: Employee |  
e.getSalary() > 100,000)

**context** Company **inv**

self.departments – **> forAll**(d: Department |  
d.members – **> exists** (e: Employee |  
e.getSalary() > 100,000))

# Constraints over collections

**context** Company **inv**

self.departments – > **exists**(d: Department |  
    d.members – > **forAll** (e: Employee | e.getSalary() > 50,000))

**context** Employee **inv**:

Employee.allInstances() – >

**forAll**(e1,e2: Employee |  
    e1.employeeId=e2.employeeId **implies** e1=e2)

# Selection of collection elements

**context** Company **inv**:

self.employees – > select(e: Employee | e.getSalary() > 50,000 )  
– >notEmpty()

**context** Company **inv**:

self.departments – >  
collect(d:Department | d.noOfEmployees)  
– >sum()==self.noOfEmployees

# Type conformance

- Subtype polymorphism
- Type conformance
- Inheritance and type conformance
- Type conformance for collection types
- Parametric types and type conformance

# Type conformance

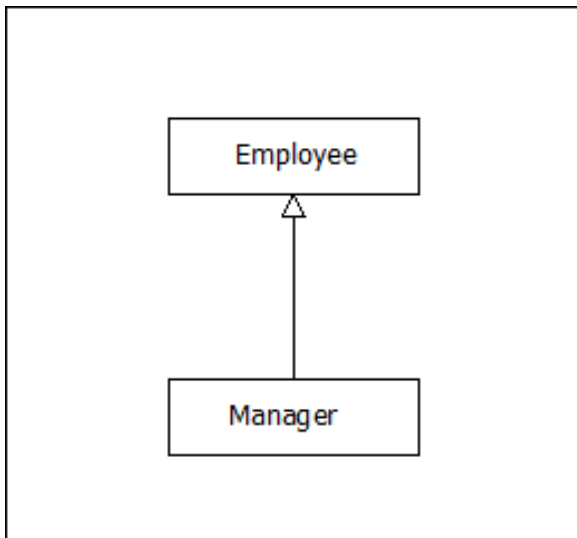


Figure: Type conformance and inheritance

# Type conformance

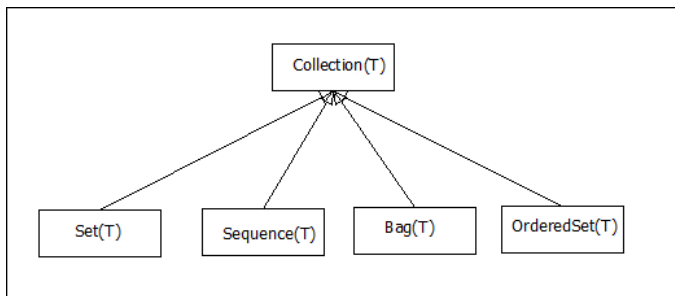


Figure: Type conformance for collections

# Type conformance

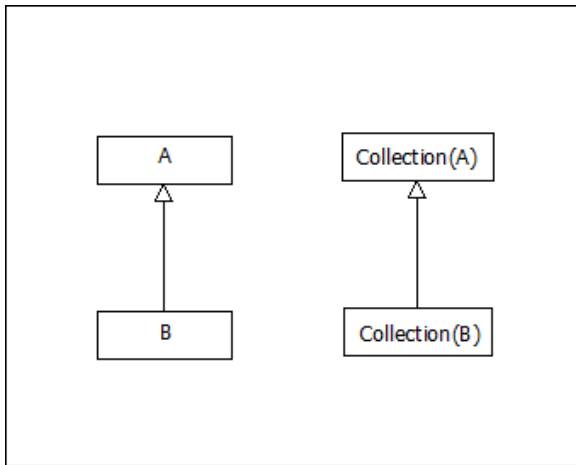


Figure: Type conformance for collections



*B* conforms to *A* implies

*Collection(B)* conforms to *Collection(A)*

*Set(B)* conforms to *Set(A)*

*Bag(B)* conforms to *Bag(A)*

*Sequence(B)* conforms to *Sequence(A)*

*OrderedSet(B)* conforms to *orderedSet(A)*.

*Set(Manager)* conforms to *Set(Employee)*

**context** Company **inv:**

self.departments – > collect(d:Department |  
d.manager) – > subset(self.employees)

**context** Manager **inv:**

Manager.allInstances() – > subset(Employee.allInstances())

- Relational queries
- Selection queries
- Selection and projection queries
- Queries on all instances

**context** Company::getDeptManagers(): Set(Manager)  
**body** self.departments – > collect(d: Department | d.manager)

**context** Company::selectWellPaid(pay: Real): Set(Employee)  
**body** self.employees – > select(e:Employee | e.getSalary() > pay)

**context** Company:: selectLargeCompanies(size: integer):  
Set(Company)  
**body** Company.allInstances – > select(c.Company |  
c.noOfEmployees > size)

# Collection(T)

Type: Collection(T)

count(obj: T): Integer

size(): Integer

**context** Collection(T)::includes(obj: T): Boolean  
**post** result = (self — > count(obj) > 0)

**context** Collection(T)::isEmpty(): Boolean  
**post** result = (self — > size() = 0)

# Operations on collections

- Operations on collections
- Operations on sets
- Operations on ordered sets
- Operations on bags
- Operations on sequences

# Set(T)

Type: Set(T)

```
context Set(T)::count(obj:T): Integer  
  post result <= 1
```

```
context Set(T)::including(obj:T): Set(T)  
  post result - > forAll(x:T | self - > includes(x) or x=obj)  
  post self - > forAll(x:T | result - > includes(x))  
  post result - > includes(obj)
```

```
context Set(T)::excluding(obj:T): Set(T)
post result – > forAll(x:T | self – > includes(x) and (x <> obj))
post self – > forAll(x:T | result – > includes(x)) = (x <> obj)
post not (result – > includes(obj))
```



```
context Set(T)::union(s: Set(T)): Set(T)
  post result — >forAll(x:T | self— >includes(x) or s— >includes(x))

context Set(T)::intersection(s: Set(T): Set(T)
  post result — >forAll(x:T | self— >includes(x) and s— >includes(x))
```

```
context Set(T):: =(s: Set(T): Boolean  
  post result – >forAll(x:T | self – >includes(x) and  
    s – > forAll (x:T | s – >includes(x))
```

# Set constraints

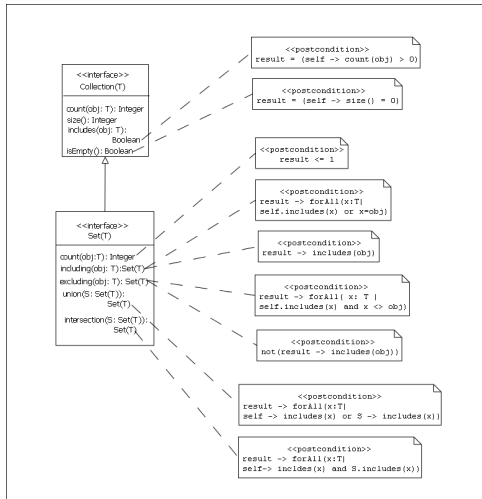


Figure: Constraints for Set(T)

# OrderedSet(T)

Type: OrderedSet(T)

**context** OrderedSet(T)::at(i: Integer): T  
**pre**  $i \geq 1$  **and**  $i \leq \text{self.size}()$

**context** OrderedSet(T)::indexOf(obj: T): Integer  
**pre**  $\text{self.includes}(\text{obj})$   
**post**  $\text{self} -> \text{at}(\text{result}) = \text{obj}$

```
context OrderedSet(T):: first(): T  
  post result = self.at(1)
```

```
context OrderedSet(T):: last(): T  
  post result = self.at(self - > size())
```

# Bag(T)

Type: Bag(T)

```
context Bag(T):: including(obj:T): Bag(T)
  post result - > forAll(x:T |
    if x=obj then
      result - > count(x) = self - > count(x) + 1
    else result - > count(x) = self - > count(x)
  endif)
```

```
context Bag(T):: excluding(obj:T): Bag(T)
  post result - > forAll(x:T |
    if x=obj then
      result - > count(x) = 0
    else result - > count(x) = self - > count(x)
  endif)
```

```
context Bag(T):: union(bag: Bag(T)): Bag(T)  
  post result- > forAll (x:T | result- >count(x) =  
    self - >count(x) + bag- >count(x))
```

```
context Bag(T):: intersection(bag: Bag(T)): Bag(T)  
  post result- > forAll (x:T | result- >count(x) =  
    self- >count(x).min( bag- >count(x)))
```

```
context Bag(T)::=(bag: Bag(T): Boolean  
  post result = (self- > forAll (x:T |  
    self- >count(x) = bag- >count(x)))
```



# Bag constraints

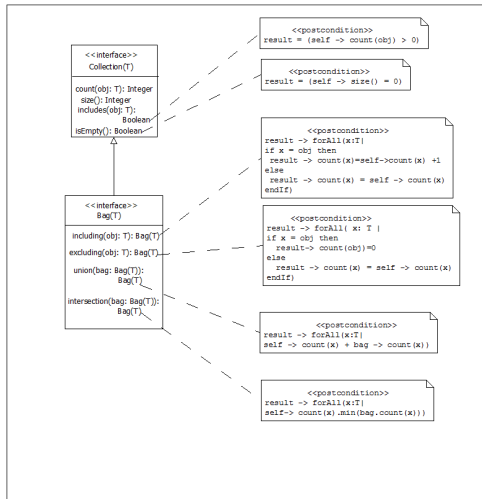


Figure: Constraints for Bag(T)

# Constraints and inheritance

- Inheritance of constraints
- Strengthening invariants
- Strengthening postconditions
- Invariance of preconditions

# Constraints and inheritance

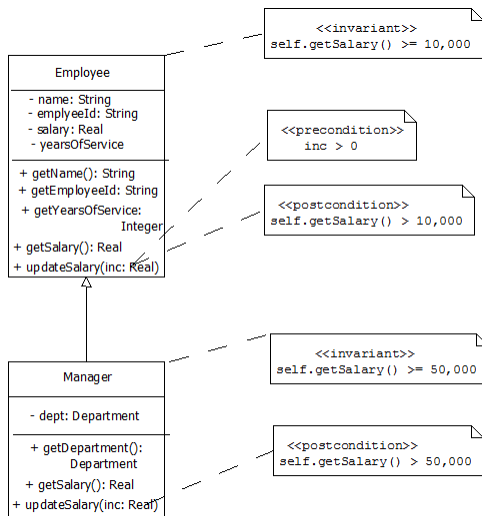


Figure: Inheritance and constraints

# Constraints and inheritance

**context** Employee **inv**  
self.getSalary()  $\geq$  10,000

**context** Manager **inv**  
self.getSalary()  $\geq$  50,000

# Constraints and inheritance

```
context Employee:: updateSalary(inc: Real)  
  pre inc > 0
```

```
context Employee:: updateSalary(inc: Real)  
  post self.getSalary() > 10,000
```

# Constraints and inheritance

```
context Manager:: updateSalary(inc: Real)  
  post self.getSalary() > 50,000
```

```
context Manager:: updateSalary(inc: Real)  
  pre self.getYearsOfService() > 2  
  post self.getSalary() > 50,000
```