Java Design By Contract

Contents

[Design by Contract 1](#_Toc68695164)

[Hauptidee 1](#_Toc68695165)

[Non-Redundancy 2](#_Toc68695166)

[Beispiel 2](#_Toc68695167)

[Quellcode 2](#_Toc68695168)

[Mit asserts 2](#_Toc68695169)

[Liskov Substitution Principle 3](#_Toc68695170)

[Folien 4](#_Toc68695171)

[Aufgaben 5](#_Toc68695172)

[WS16/17 5](#_Toc68695173)

[SS17 7](#_Toc68695174)

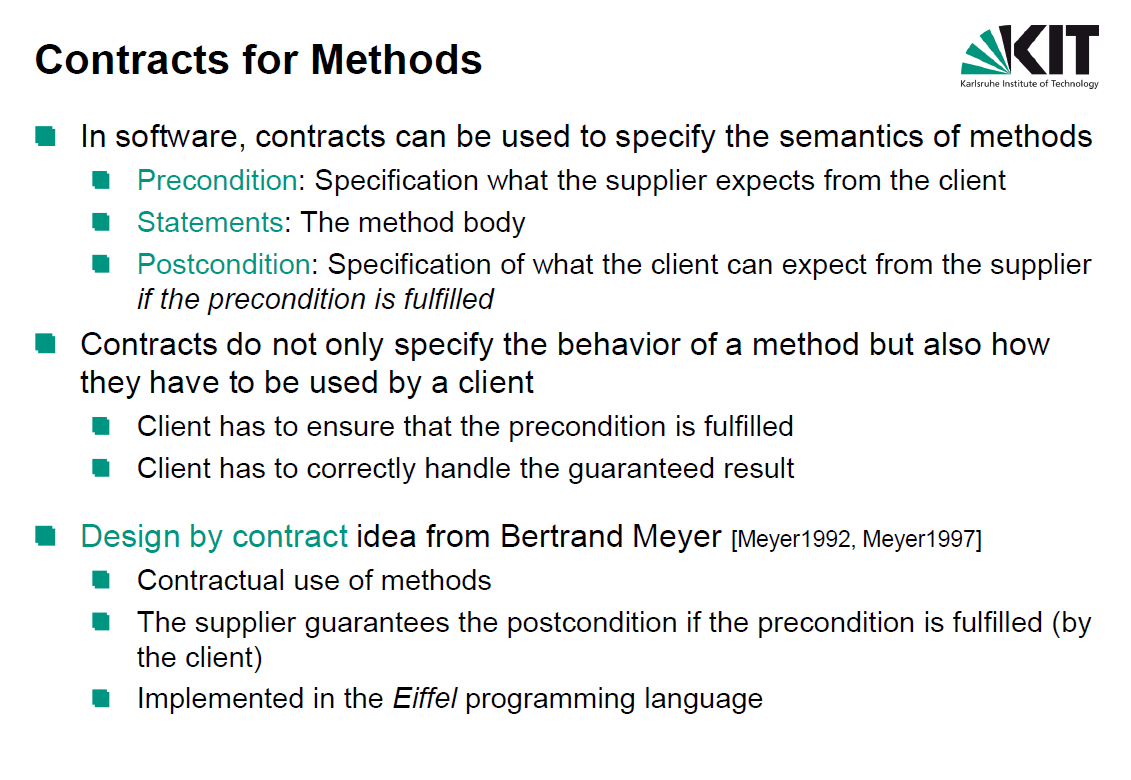
# Design by Contract

## Hauptidee

**The supplier guarantees the postcondition if the precondition is fulfilled (by the client)**

Aufrufer == Customer

Aufrufende == Supplier



## Non-Redundancy

The body of a routine shall not test for the routine’s precondition.

Beispiel

### Quellcode

public class Employee {

private boolean isEmployed;

public Employee() {

this.isEmployed = false;

}

protected void hire() {

this.isEmployed = true;

}

protected void fire() {

this.isEmployed = false;

}

public boolean isEmployed() {

return isEmployed;

}

}

### Mit asserts

1. Alle Vorbedingungen (Null, Zustand, alte Liste speichern)

2. Aktion

3. Alle Nachbedingungen

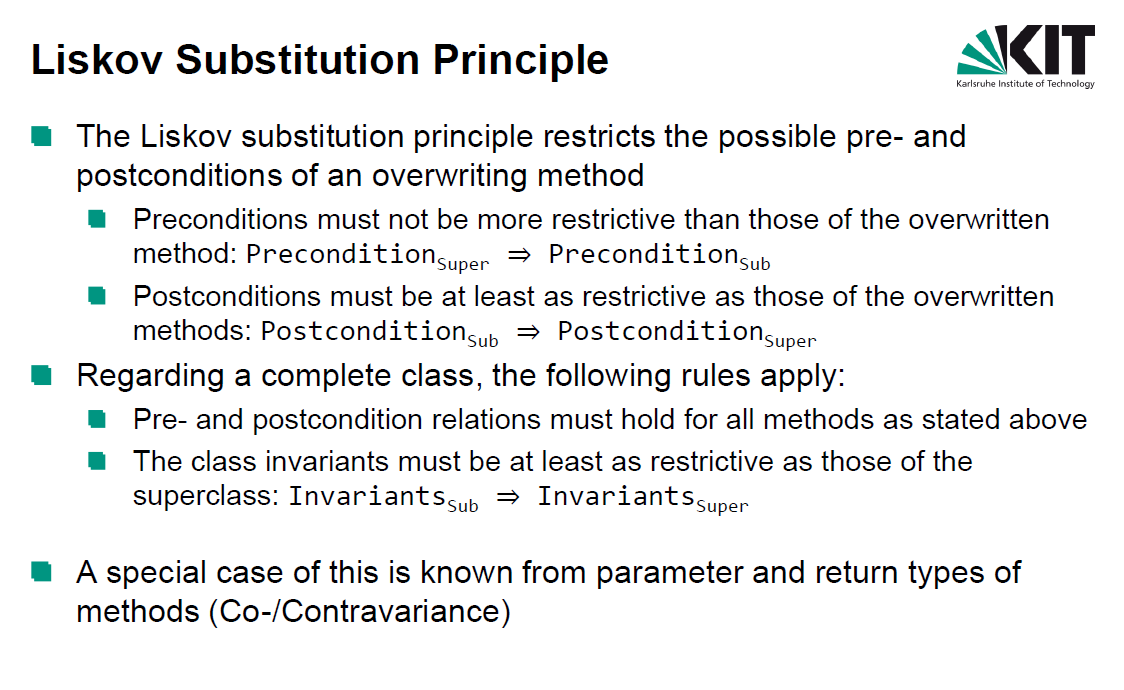
@Override  
public void hire(Employee employee) {  
 assert ( employee!= null);  
 assert (!employees.contains(employee));  
 assert (!employee.isEmployed());  
 List<Employee> oldEmployees = new ArrayList<>(employees);  
  
 employee.hire();  
 employees.add(employee);  
  
 assert (employees.contains(employee));  
 assert (employee.isEmployed());  
 assert (employees.containsAll(oldEmployees));  
 assert (employees.size() == oldEmployees.size() + 1);  
  
}  
  
@Override  
public void fire(Employee employee) {  
 assert (employee!= null);  
 assert (employees.contains(employee));  
 assert (employee.isEmployed());  
 List<Employee> oldEmployees = new ArrayList<>(employees);  
  
 employee.fire();  
 employees.remove(employee);  
  
 assert (!employees.contains(employee));  
 assert (!employee.isEmployed());  
 assert (oldEmployees.containsAll(employees));  
 assert (employees.size() == oldEmployees.size() - 1);  
}

## Liskov Substitution Principle

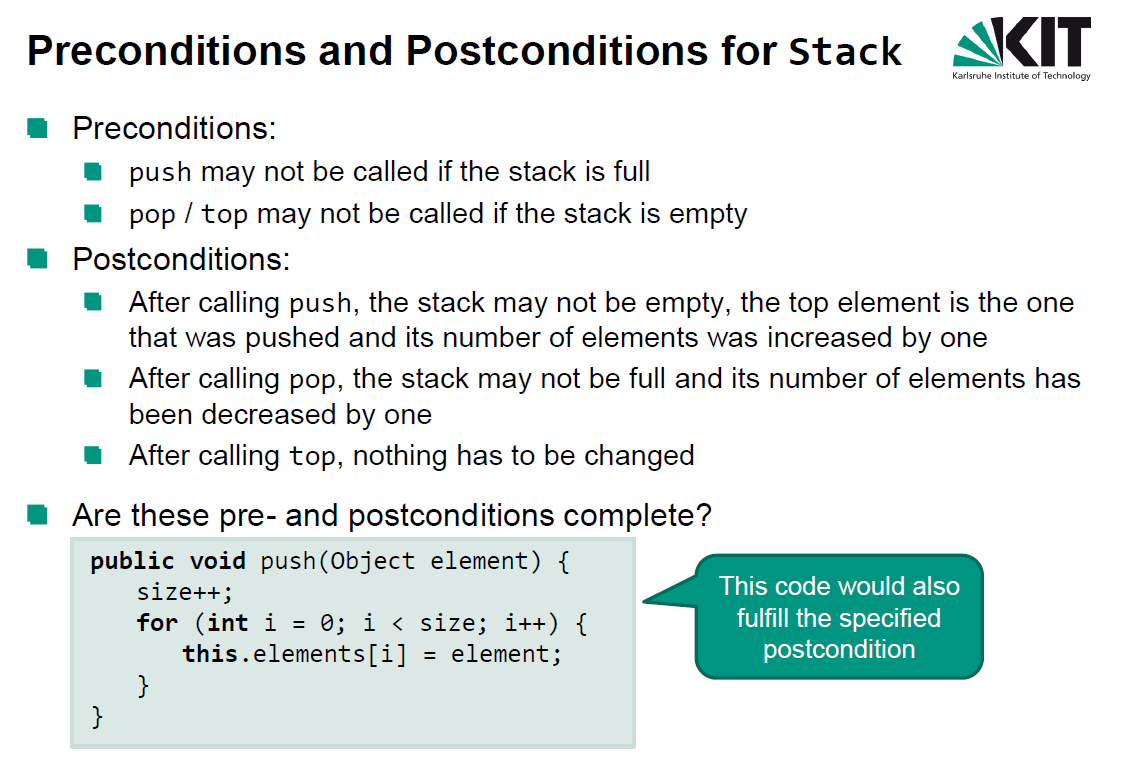
Class A extends Class B and overwrites the method foo()

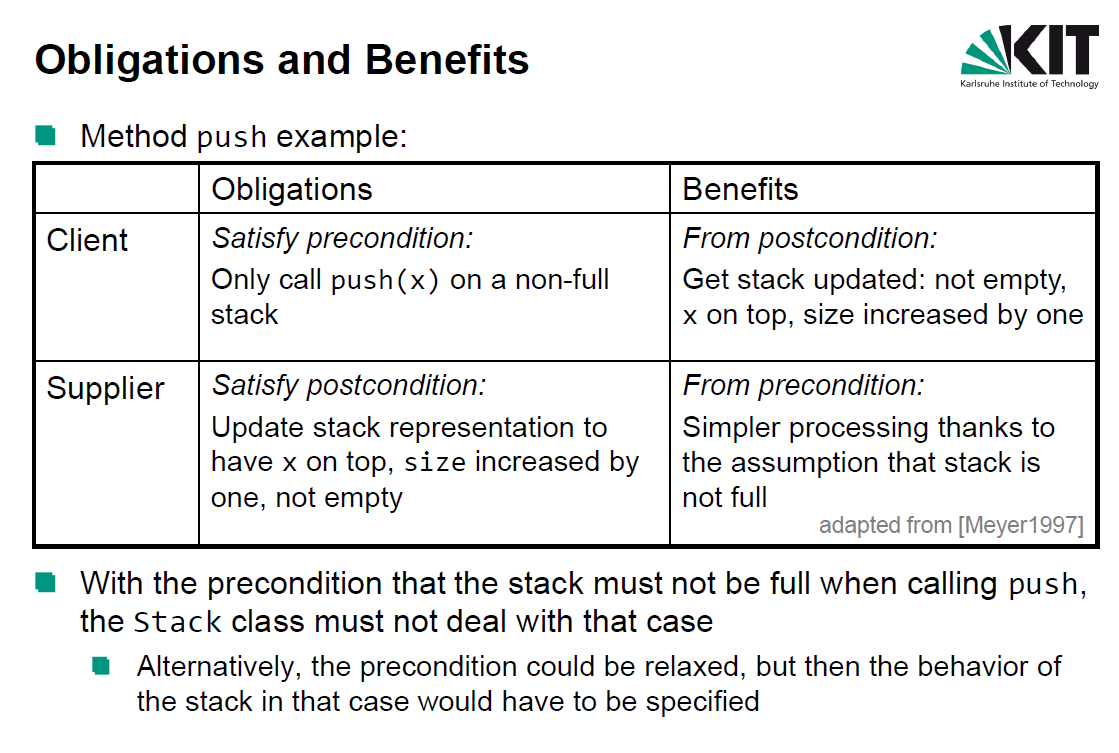
A.foo() erfüllt LSP genau dann, wenn:

1. **Preconditions** von A.foo() sind **nicht strickter** als bei B.foo()
2. **Postconditions** von A.foo() sind **nicht schwächer** als bei B.foo()



## Folien

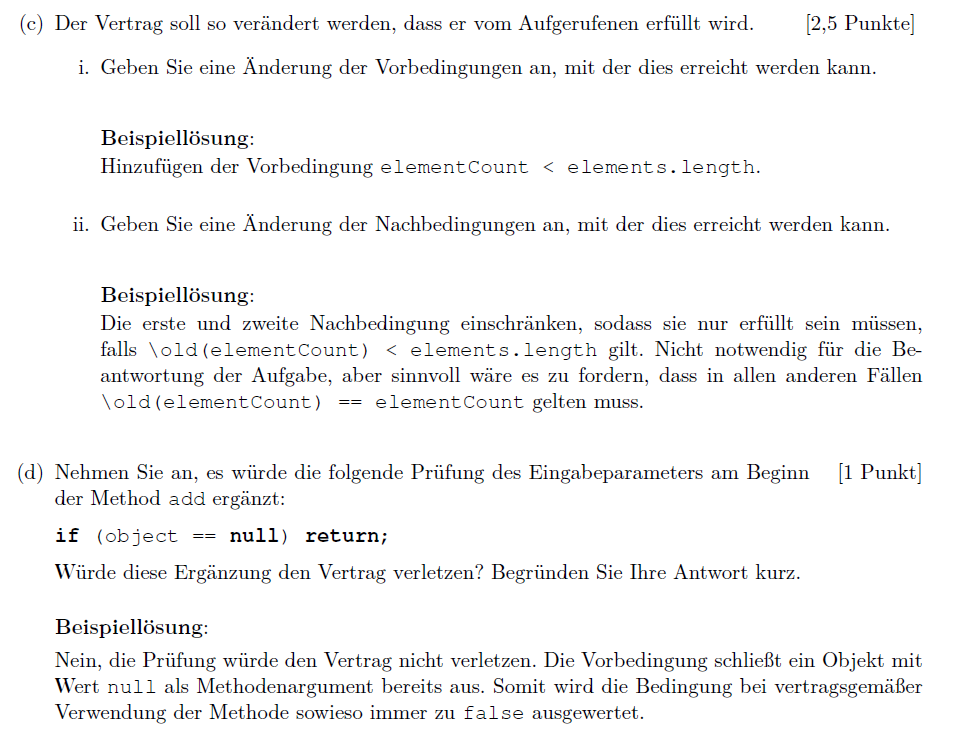
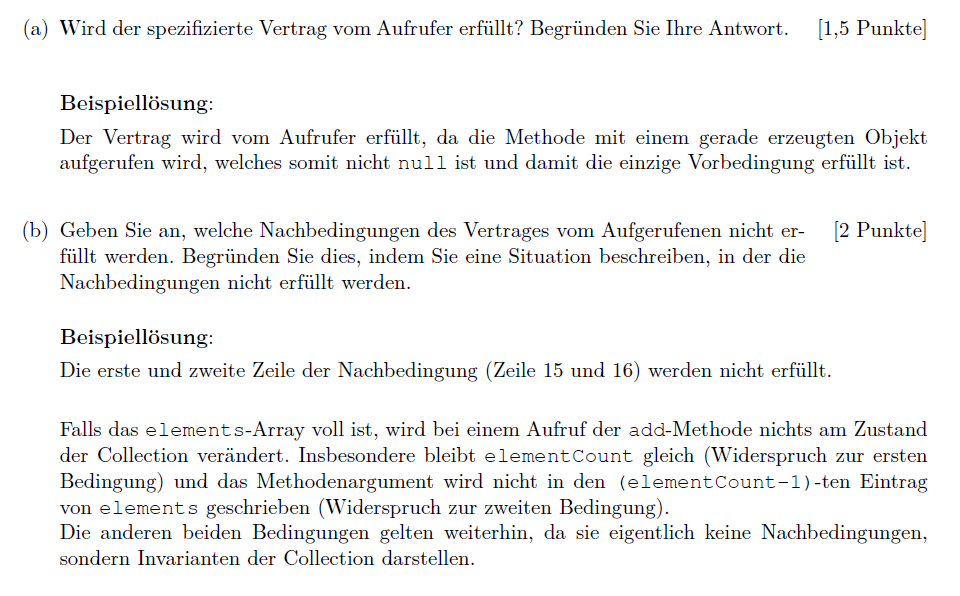




# Aufgaben

## WS16/17





## SS17

**Set** kann keine Duplicate haben!

