Object World Berlin '99
Design & Components
Berlin, May 17th-20th 1999

Examples of Design by Contract in Java

using Contract, the Design by Contract™ Tool for Java™

Reto Kramer, kramer@acm.org

Design by Contract - What is it ?

- **Classes of a system communicate with one another on the basis of precisely defined benefits and obligations.**

  [Bertrand Meyer, CACM, Vol. 36, No 9, 1992]

Example - Class Person

```
/**
 * @invariant age_ > 0
 */
class Person {
 protected age_;

 /**
  * @post return > 0
  */
 int getAge() {..}

 /**
  * @pre age > 0
  */
 void setAge( int age ){..}

…}
```

- New comment-tags: @**pre**, @**post**, @**invariant**
- All instances of person must have a positive age.
- Clients are promised that the age is positive provided that:
- Clients are obligated to pass positive ages only. Service will be denied otherwise.

Meaning of pre, post and invariant

- If **preconditions** are not obeyed by the **client** of the class' method, the service provider will deny its service !
- If any **postcondition** is violated, it uncovers a problem on the **service provider** side.
- If any class **invariant** is violated it uncovers a problem on the **service provider** side.
- The problem can be
  - implementation error
  - not specific enough preconditions

Benefits - Obligations

	Benefit	Obligation
Client	**- no need to check output values - result guaranteed to comply to postcondition** ④	**satisfy pre-conditions** ①
Provider	② **- no need to check input values - input guaranteed to comply to precondition**	③ **satisfy post-conditions**

Cambridge Technology Partners

So, is it like “**`assert.h`**” ?

- Assert statements are a great tool - design by contract even goes one step beyond them:
  - assert does not provide a contract
  - clients can not see asserts as part of the interface
  - does not have a semantic associated with it
  - not explicit whether they represent pre-, post-conditions or invariants
  - no OO support (e.g. inheritance), see later
  - does not lead to “automatic” documentation

Example - A Simple Interface

```
 1:  interface Person {
 2:
 3:    /**age always positive
 4:     * @post return > 0
 5:     */
 6:    int getAge();
 7:
 8:    /** age always positive
 9:     * @pre age > 0
10:     */
11:   void setAge( int age );
12: }
```

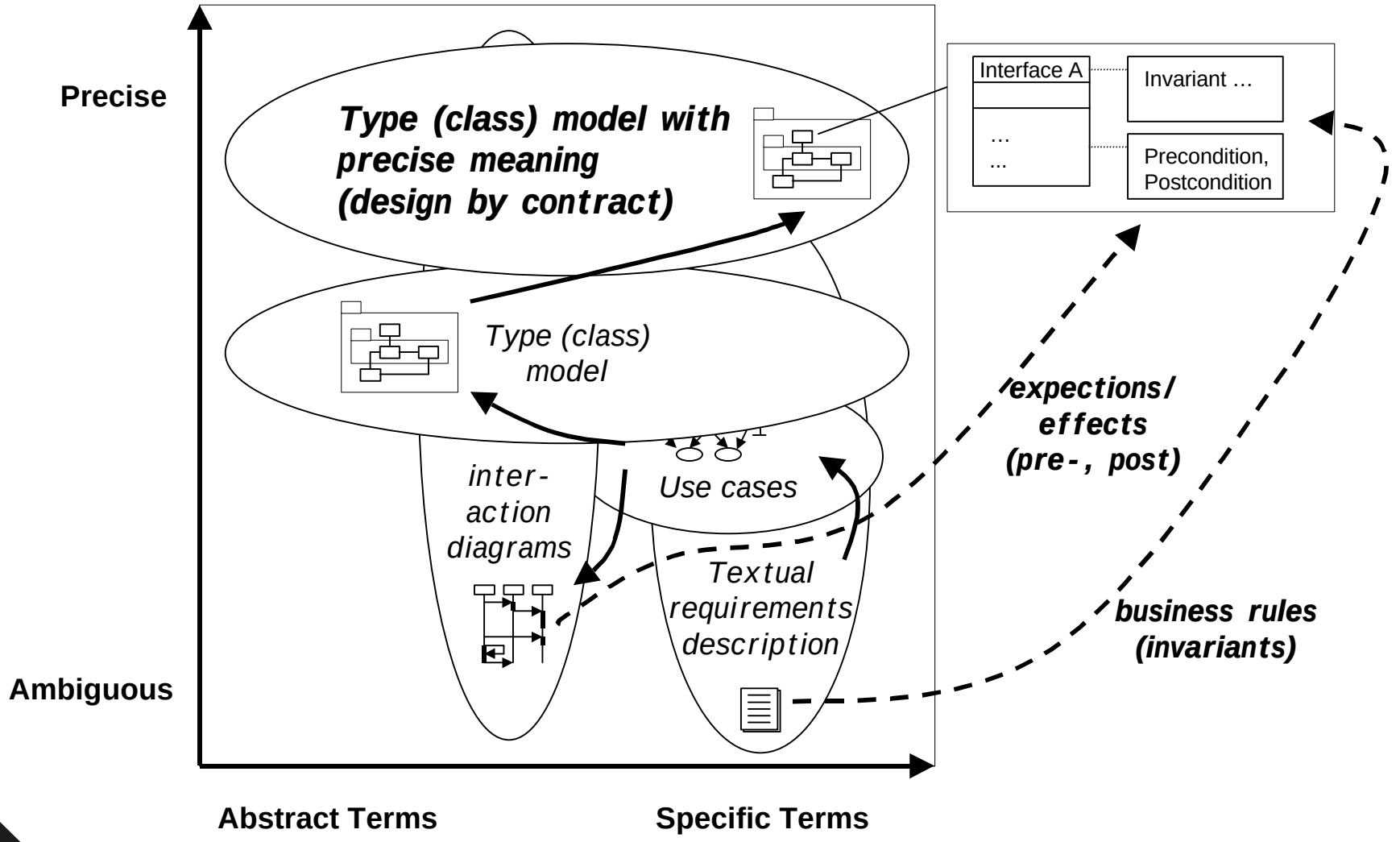
```
 1: class Employee implements Person {
 2:
 3:    protected int age_;
 4:
 5:    public int getAge() {
 6:      return age_;
 7:    };
 8:
 9:    public void setAge( int age ) {
10:     age_ = age;
11:   };
12: }
```

Cambridge Technology Partners

Benefits - General

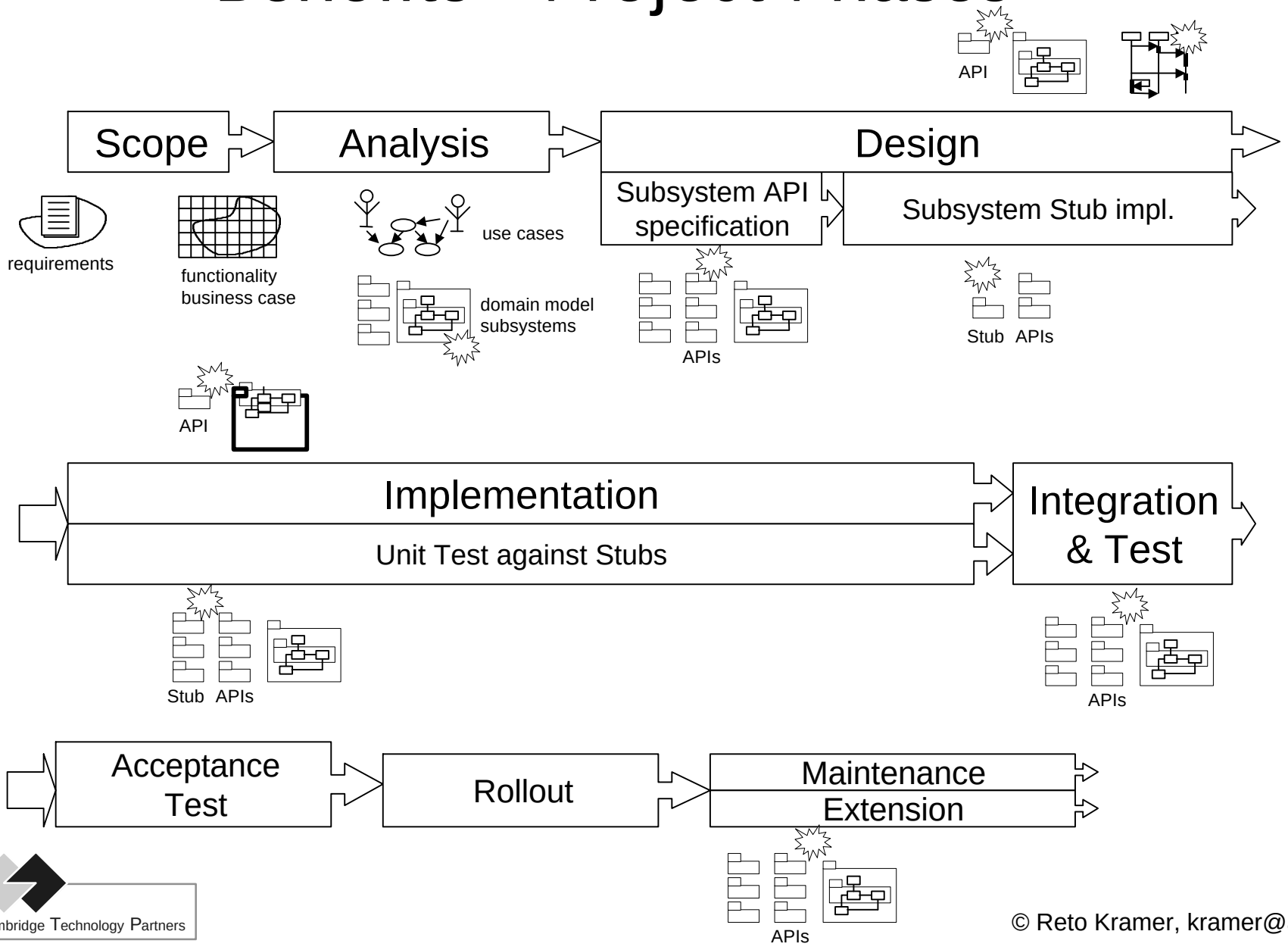
- failures occur close to the faults
  (I.e. during integration tests and field use!)
- interface documentation always up-to-date, can be trusted!
- documentation can be generated automatically (iDoclet)
- contract specification serves as a basis for black box testing of classes (test-driver spec)

Benefits - Abstraction and Precision



Cambridge Technology Partners

Benefits - Project Phases



Benefits - Project Roles

- **Class user**
  - postconditions guaranteed
  - can trust documentation
- **Class provider**
  - preconditions guaranteed
  - automatic documentation
- **Test manager**
  - more accurate test-effort estimation
  - black box spec for free
- **Project manager**
  - easier to preserve design over a long time
  - reduced maintenance effort in the long run (failure close to fault)
  - enables unambiguous interface specification
  - lower documentation cost
  - fearless reuse (enables specification of reusable classes)

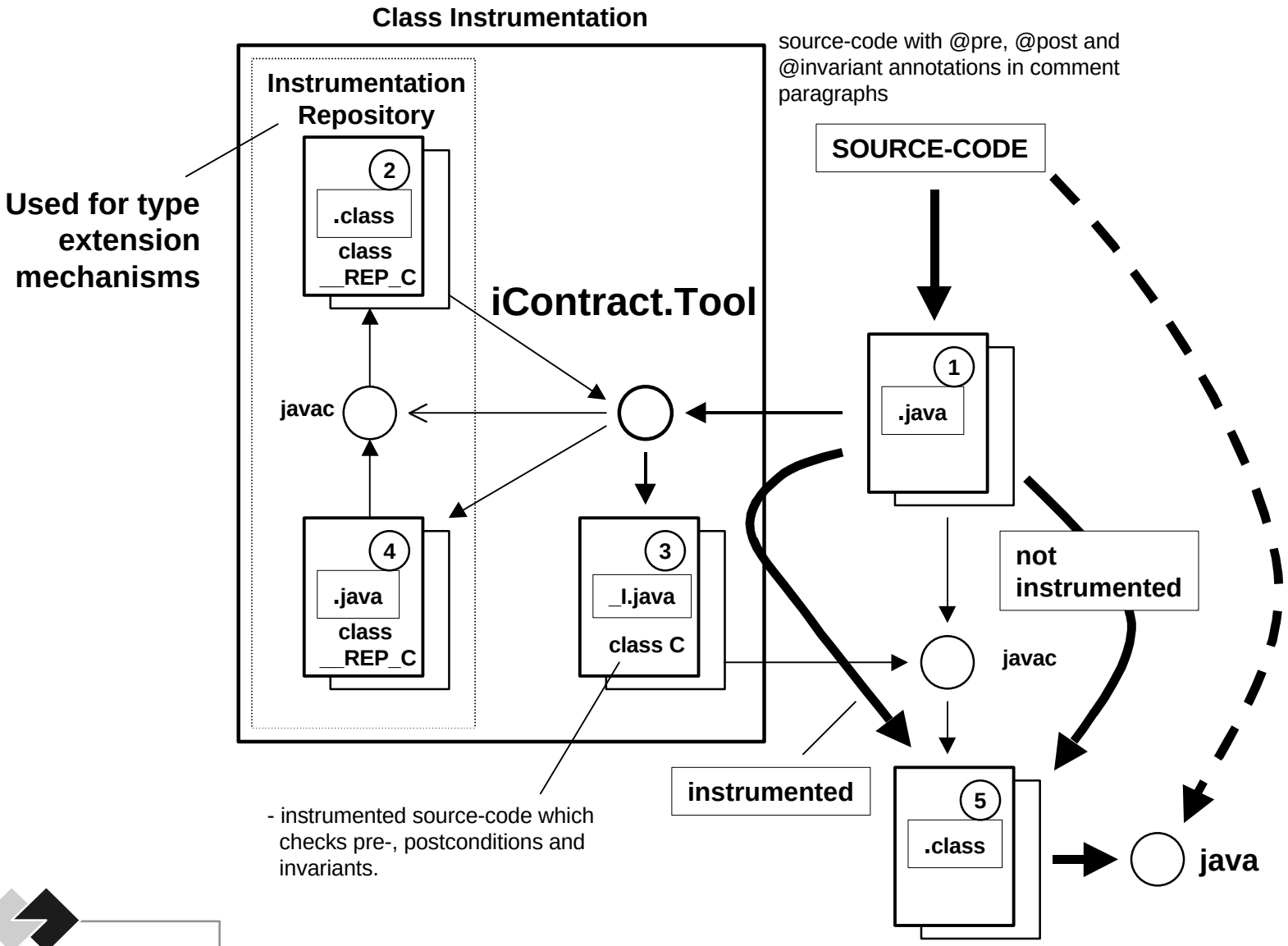
References

- iContract: http://www.reliable-systems.com
- Books:
  - “Object Oriented Software Construction”, 2nd edition, Bertrand Meyer, Prentice Hall, 1997
  - “Objects, Components and Frameworks with UML”, D.F. D’Souza, A. Cameron Wills, Addison Wesley, 1999
- Eiffel [Interactive Software Engineering, ISE] http://www.eiffel.com
- UML 1.1 / Object Constraint Language (OCL) http://www.rational.com

iContract - the Tool

- source code pre-processor
- no run-time library required
- compatible with OCL
  - old value, x@pre
  - return value
  - quantifiers: forall, exists
- supports Java type extension mechanisms (contract propagation)

Tool Components



Cambridge Technology Partners

Performance Tuning

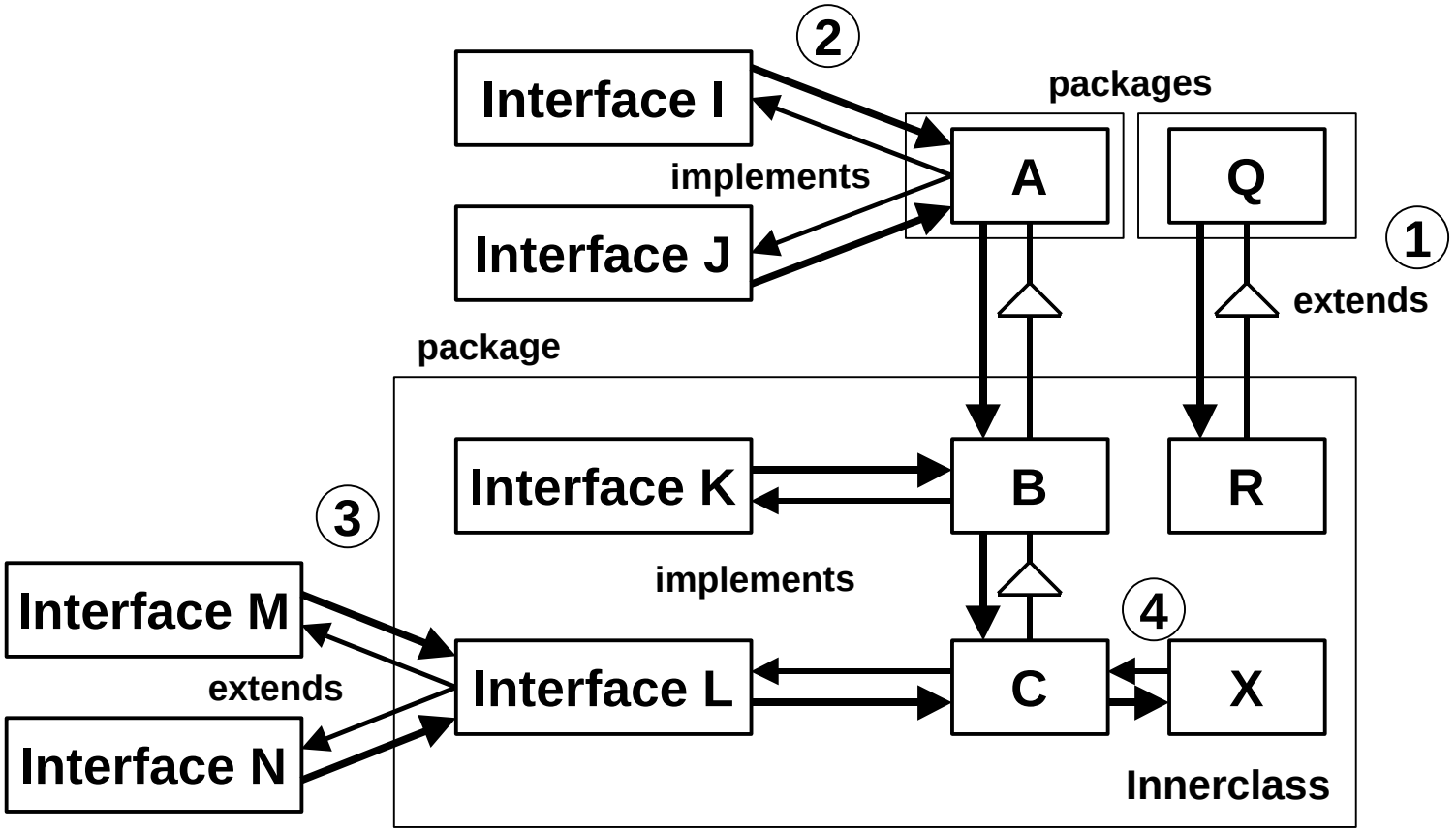
- Check instrumentation is done per .java file (public class)
- Performance critical classes can be excluded from the checks

- Files can be instrumented with any combination of checks for:
  - pre-
  - post-conditions and
  - invariants

- E.g. if implementation is tested thoroughly, only check preconditions

Cambridge Technology Partners

Java Language Support



Cambridge Technology Partners

Java Language Support (con't)

- All but private methods are instrumented with invariant checks.
- The finalize() method is not instrumented with invariant checks.
- Invariant checks are "**`synchronized`**"
- Recursive invariant checks are avoided automatically
- Default constructors are added to classes automatically, if needed
- In constructors the delegation to this(...) and super(...) is put in front of the precondition check (javac demands this).

Specification Language

- Propositional logic with quantifiers
- Any expression that may appear in an if(...) condition may appear in a pre-, post- and invariant expression
- Scope:
  - as if the invariant were a method of the class, interface
  - as if the pre- and postcondition were are statement of the method

Specification Language (con't)

- ***forall*** Type t **in** <enumeration> **|** <expr>
  - <collection**>->forAll**(t **|** <expr>)
- ***exists*** Type t **in** <enumeration> **|** <expr>
  - <collection>**->exists**(t **|** <expr>)
- <a> ***implies*** <b> (same as OCL)
  - same as OCL

- Differences between iContract and OCL
  - syntactic & iContract needs to know Type!

Specification Language (con't)

- In postconditions references to the following pseudo-variables are allowed:

- *return* denotes the return value of a method
  - this is called "result" in OCL

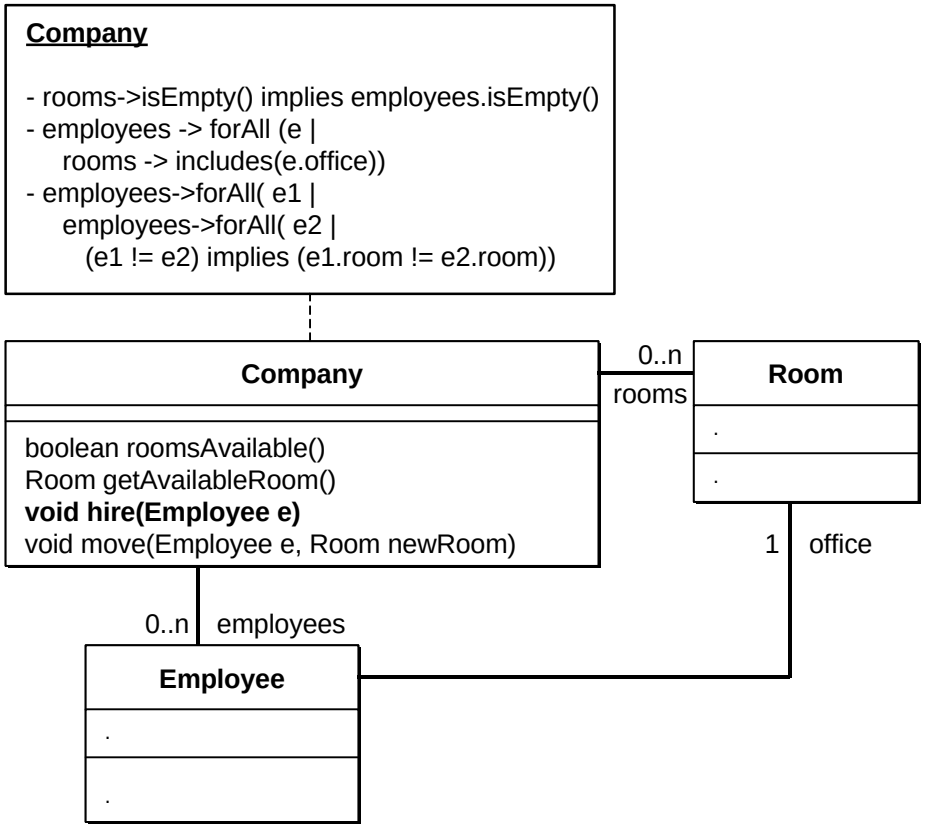
- *<expression>@pre* denotes the value of the expression (e.g. variable) when the method was entered - notation from UML / OCL "old value reference"
  - same as OCL

Example

- Office Management System
  - Manage the rooms available to a company. Provide new hires with office and support employees that move from one office to another.
- Focus on
  - Initial type model of domain (UML)
  - Add business constraints and rules (OCL)
  - Add precise meaning of operations (OCL)
  - Generate Java (iContract)

Office Management Example (hire)

1



2

void Company:hire(Employee e)

pre: (e != null) && (!employees->includes(e))

post:
- employees->includes(e)

- getAvailableRoom()@pre != getAvailableRoom()
  // hire must call an unspecified method that will
  // ensure that a new, available room is choosen

- e.office == getAvailableRoom() // SIDE EFFECT FREE!

Room Company:getAvailableRoom()

pre: roomsAvailable()
post:
- result != null
- rooms->includes(result)
- !employees->exists(e | e.office == result)
- result == getAvailableRoom() // SIDE EFFECT FREE!

boolean Company:roomsAvailable()

pre: TRUE
post: result == rooms->exists(room |
!employees->exists(e |
e.office == room))

Cambridge Technology Partners

Office Management Example (move)

3

void Company:move(Employee e, Room newRoom)

pre:
- (e != null) && (newRoom != null)
- employees->includes(e)
- !employee->exists(e | e.office == newRoom)
  // newRoom is not anyones office

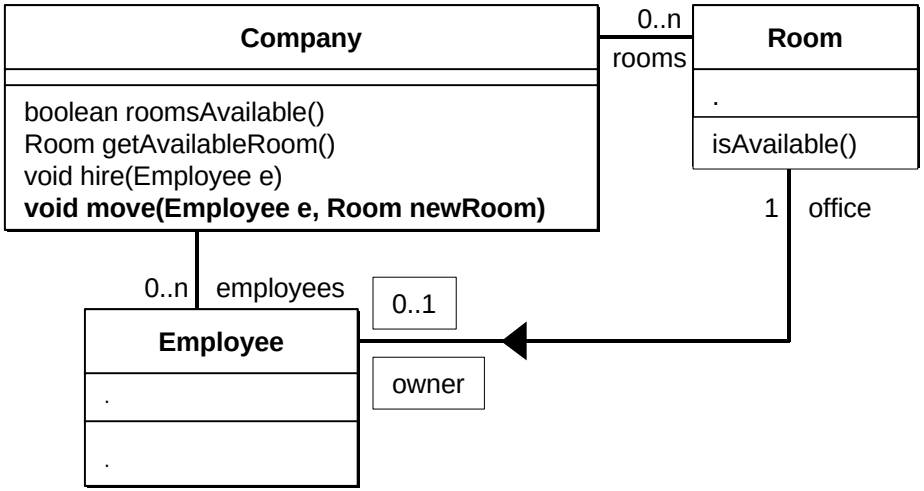
post:
- e.office == newRoom
- !employee->exists(other | other.office == e.office@pre)
  // the employee's (e) old office (e.office@pre) is not
  // used by any other employee

Room.isAavailable() ??

boolean Room:isAvailable()

pre: TRUE
post:
- result == onwer != null

4



5

void Company:move(Employee e, Room newRoom)

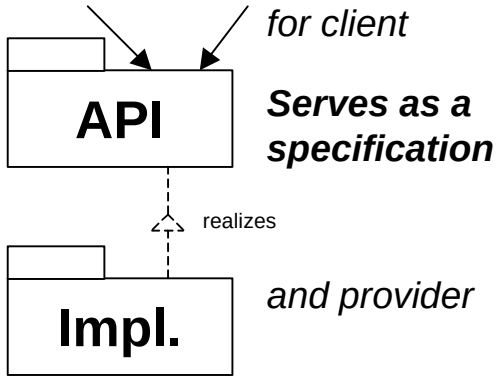
pre:
- (e != null) && (newRoom != null)
- employees->includes(e)
- newRoom.isAvailable()

post:
- e.office == newRoom
- e.office@pre.isAvailable()
- !newRoom.isAvailable()

Cambridge Technology Partners

API Specification for Subsystem

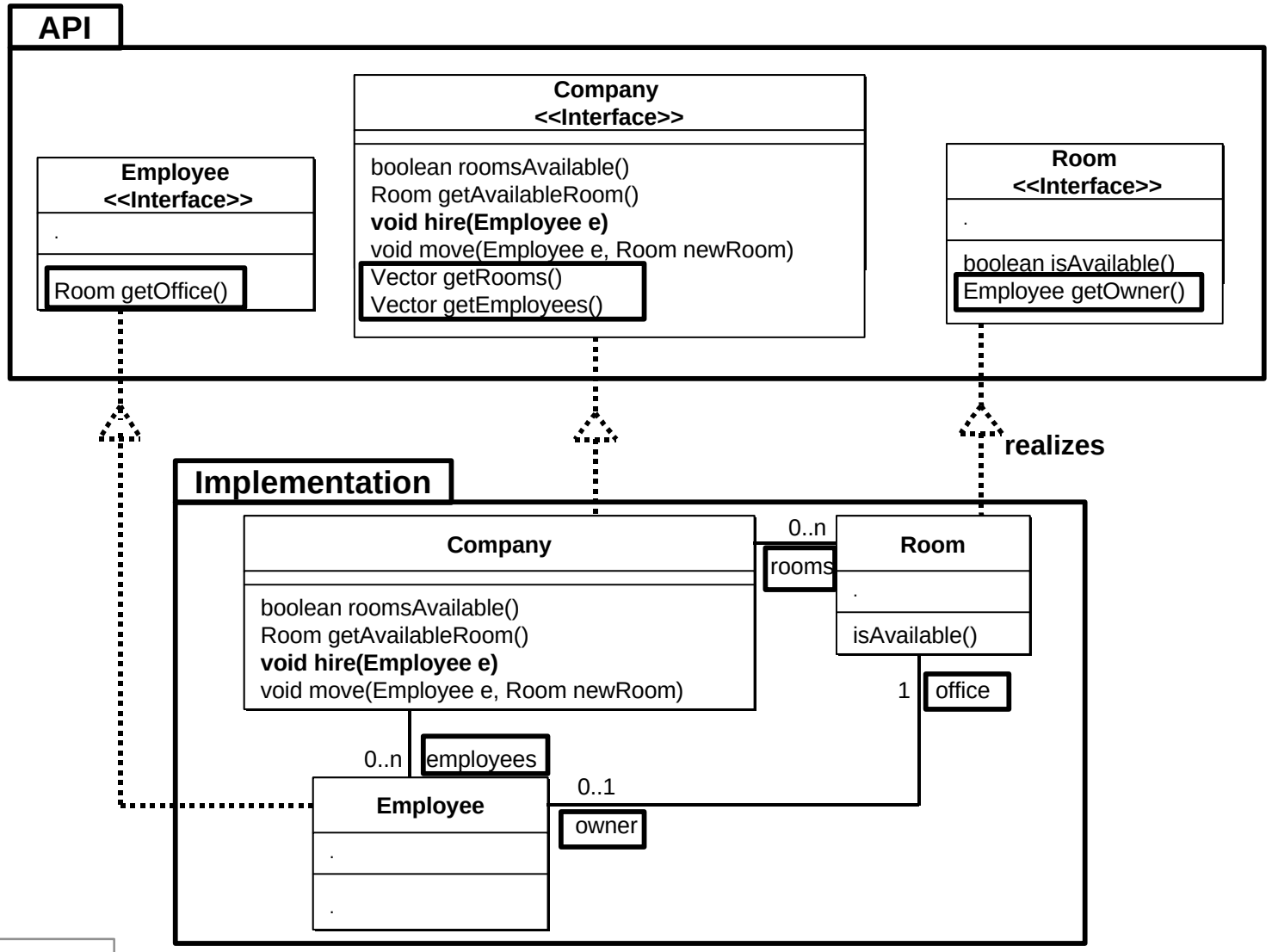
- Assume Office Management System to be a subsystem of a larger, total solution
- Hence requires proper separation of interface from implementation.



- Specification of previous slides is mapped to Java package containing interfaces.
- but what about the associations ?
- Need to create “get” methods for each role in an association …

Cambridge Technology Partners

API Specification for Subsystem



Cambridge Technology Partners

iContract

Company

- rooms->isEmpty() implies employees.isEmpty()
- employees -> forAll (e |
  rooms -> includes(e.office))
- employees->forAll(e1 |
  employees->forAll(e2 |
  (e1 != e2) implies (e1.room != e2.room))

```
/**
 * @invariant getRooms().isEmpty() implies getEmployees.isEmpty()
 * @invariant forall Employee e in getEmployees().elements() |
 *              getRooms.contains(e.getOffice())
 * @invariant forall Employee e1 in getEmployees().elements() |
 *              forall Employee e2 in getEmployees().elements() |
 *                (e1!=e2) implies (e1.getOffice()!=e2.getOffice())
 */
interface Company {
  /**
   * @pre  (e!=null) && (newRoom!=null)
   * ..
   * @post e.getOffice()@pre.isAvailable()
   */
  void move(Employee e, Room newRoom);
  ..
}
```

void Company:move(Employee e, Room newRoom)

pre:
- (e != null) && (newRoom != null)
- employees->includes(e)
- newRoom.isAvailable()

post:
- e.office == newRoom
- e.office@pre.isAvailable()
- !newRoom.isAvailable()

iContract propagates API specification into implementing classes !

Cambridge Technology Partners