Object Design: Interface Specification

Introduction into Software Engineering
Lecture 15

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

Lecture Plan

- Specifying Interfaces (Chapter 9)
  - Object Design Activities Visibilities and Information Hiding, Contracts
- Mapping Models to Java Code (Chapter 10)
  - Optimizations to address performance requirements
  - Implementation of class model components
    - Realization of associations
    - Realization of contracts
- Mapping Models to Relational Schema (Ch 10.4.4)
  - Realizing entity objects
  - Mapping the object model to a storage schema
  - Mapping class diagrams to tables.

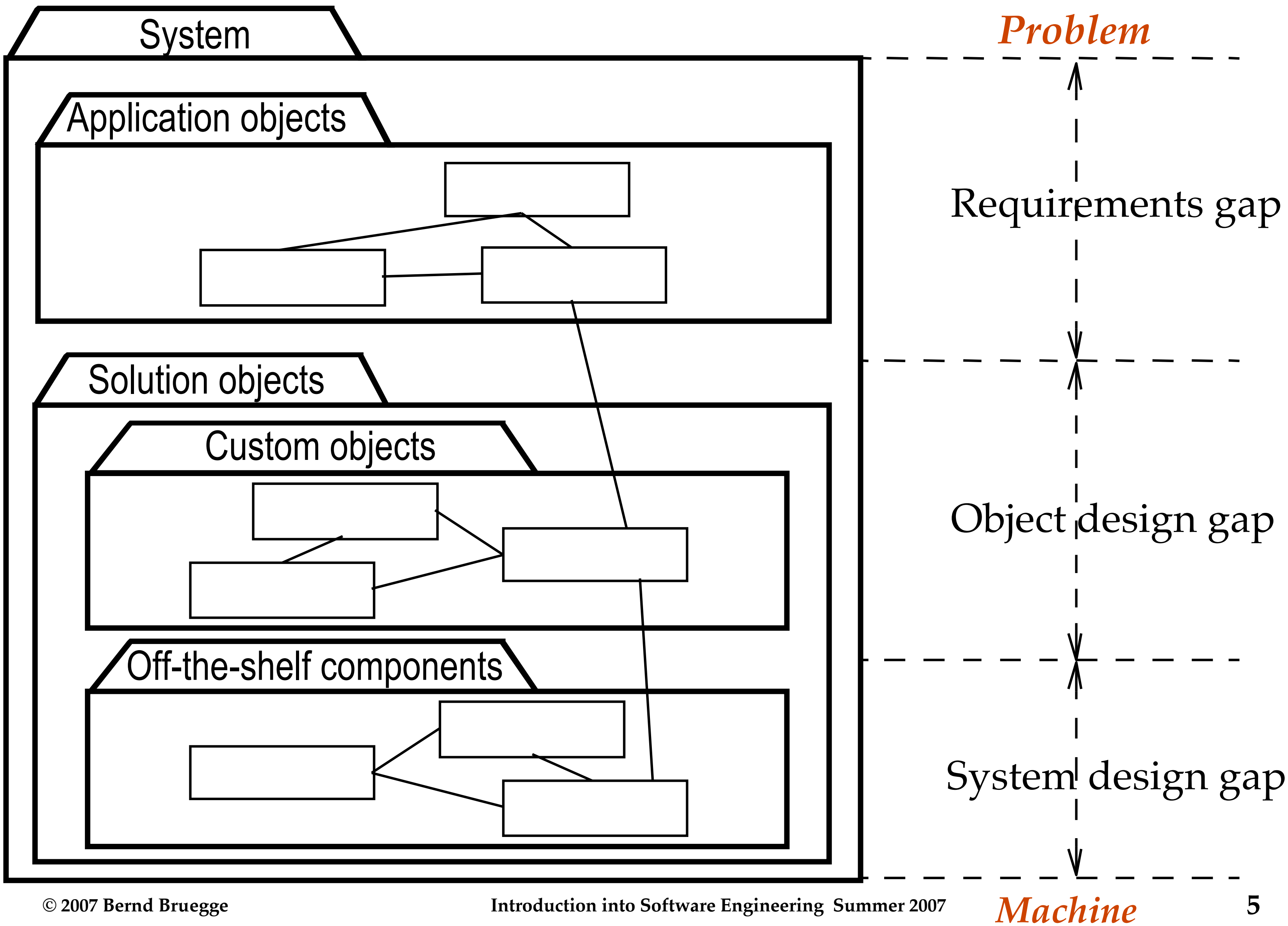
Outline of Today's Lecture

- Object Design Activities
- Visibilities
- Information Hiding
- Contracts

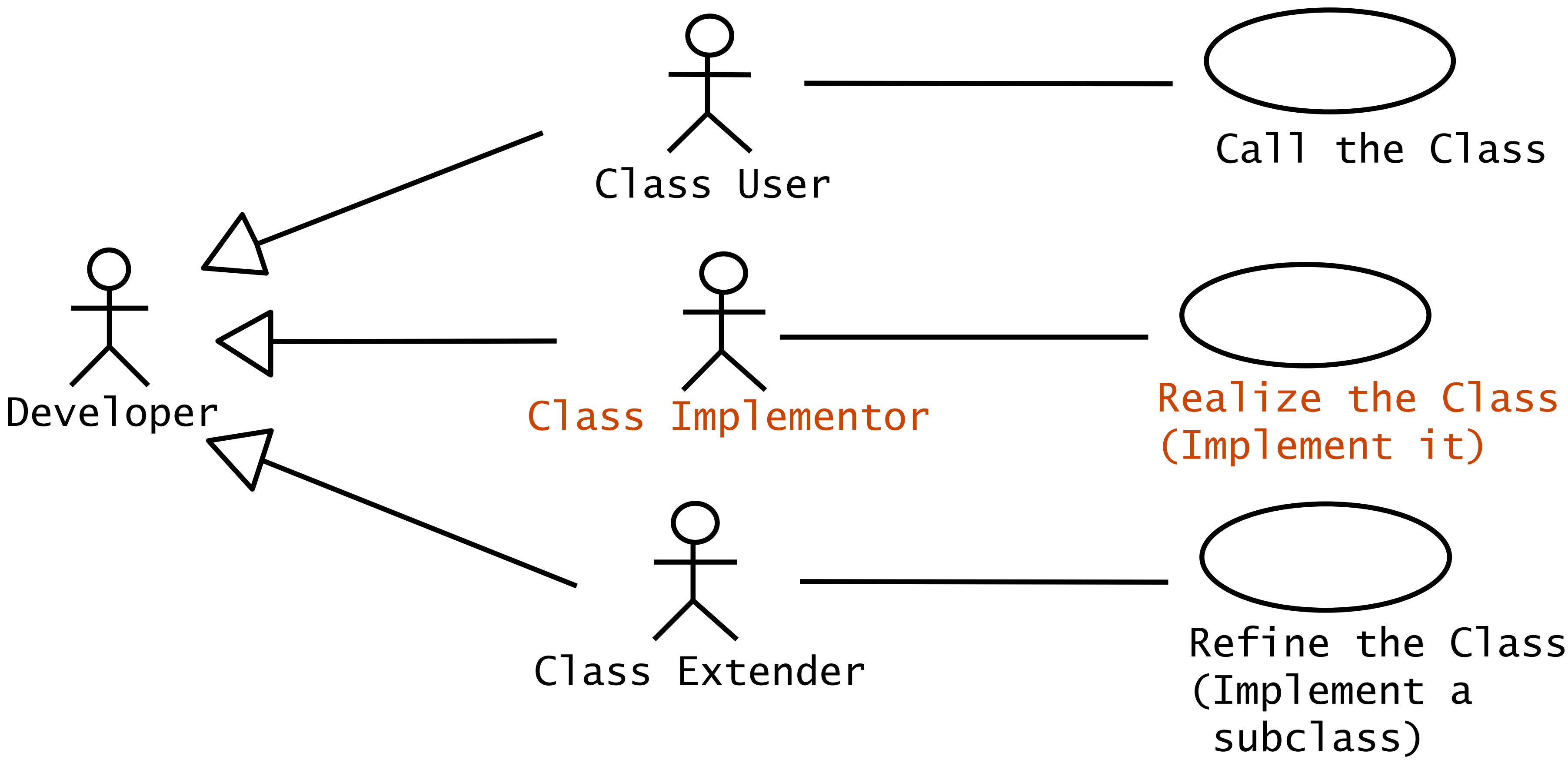
Requirements Analysis vs. Object Design

- Requirements Analysis: The functional model and the dynamic model deliver operations for the object model
- Object Design: Decide where to put these operations in the object model
  - Object design is the process of
    - adding details to the requirements analysis
    - making implementation decisions
- Thus, object design serves as the basis of implementation
  - The object designer can choose among different ways to implement the system model obtained during requirements analysis.

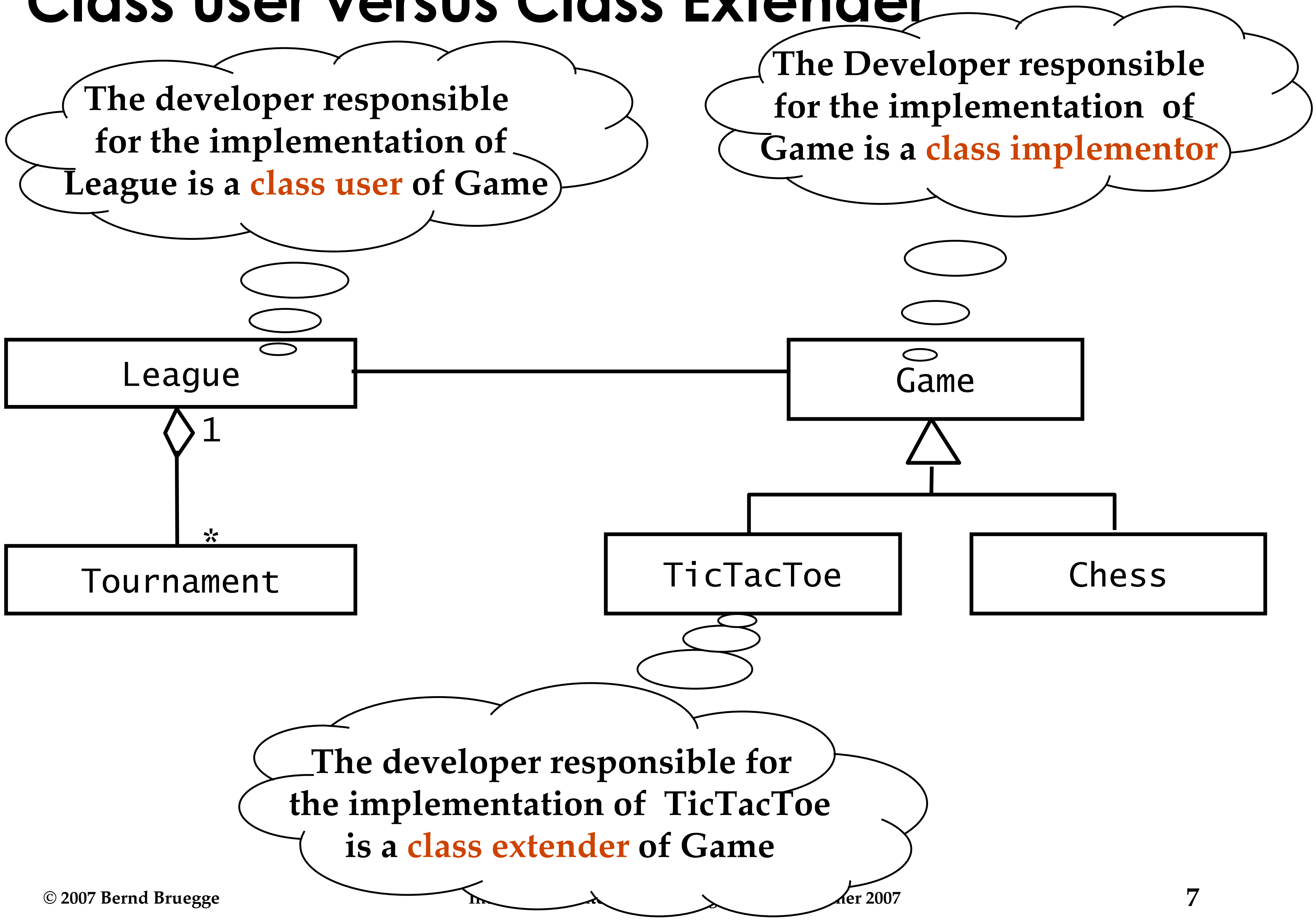
Object Design: Closing the Final Gap



Developers play 3 different Roles during Object Design of a Class



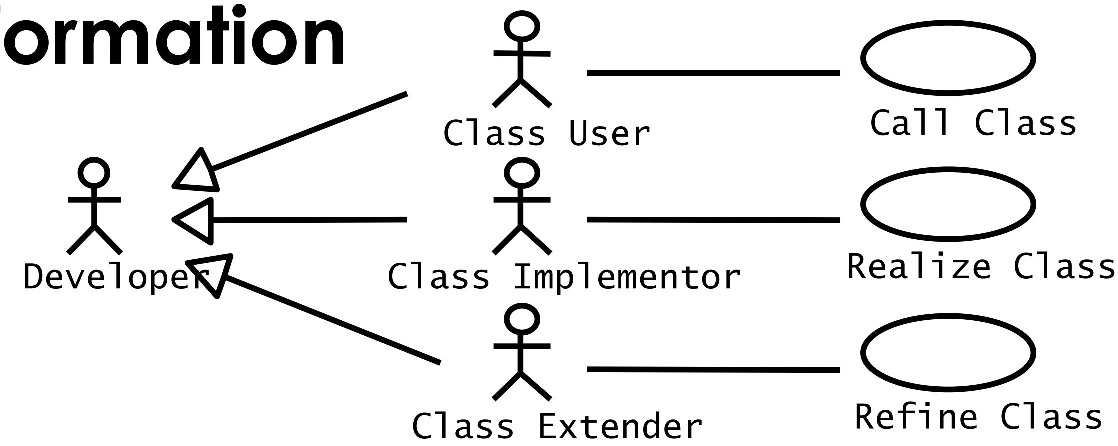
Class user versus Class Extender



Specifying Interfaces

- Requirements analysis activities
  - Identify attributes and operations without specifying their types or their parameters
- Object design activities
  - Add visibility information
  - Add type signature information
  - Add contracts.

Add Visibility Information



Class user ("Public"): +

- Public attributes/operation can be accessed by any class

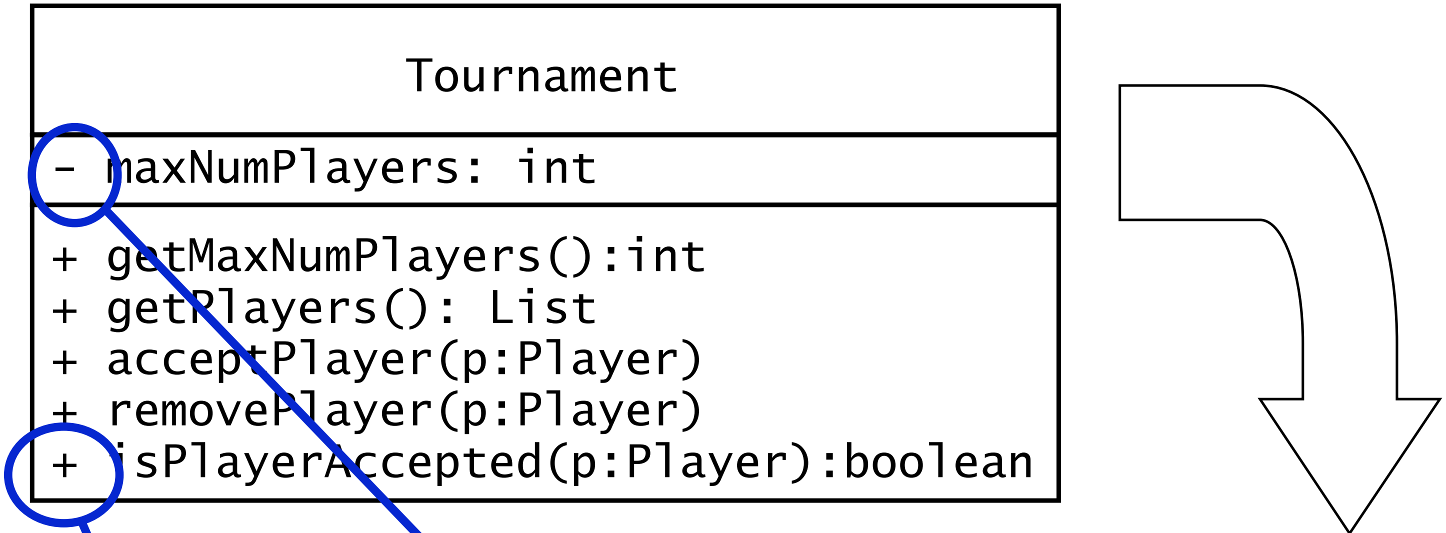
Class implementor ("Private"): -

- Private attributes and operations can be accessed only by the class in which they are defined
- They cannot be accessed by subclasses or other classes

Class extender ("Protected"): #

- Protected attributes/operations can be accessed by the class in which they are defined and by any descendent of the class.

Implementation of UML Visibility in Java




```
public class Tournament {
        private int maxNumPlayers;
public Tournament(League l, int maxNumPlayers)
public int getMaxNumPlayers() {…};
public List getPlayers() {…};
public void acceptPlayer(Player p) {…};
public void removePlayer(Player p) {…};
public boolean isPlayerAccepted(Player p) {…};
```

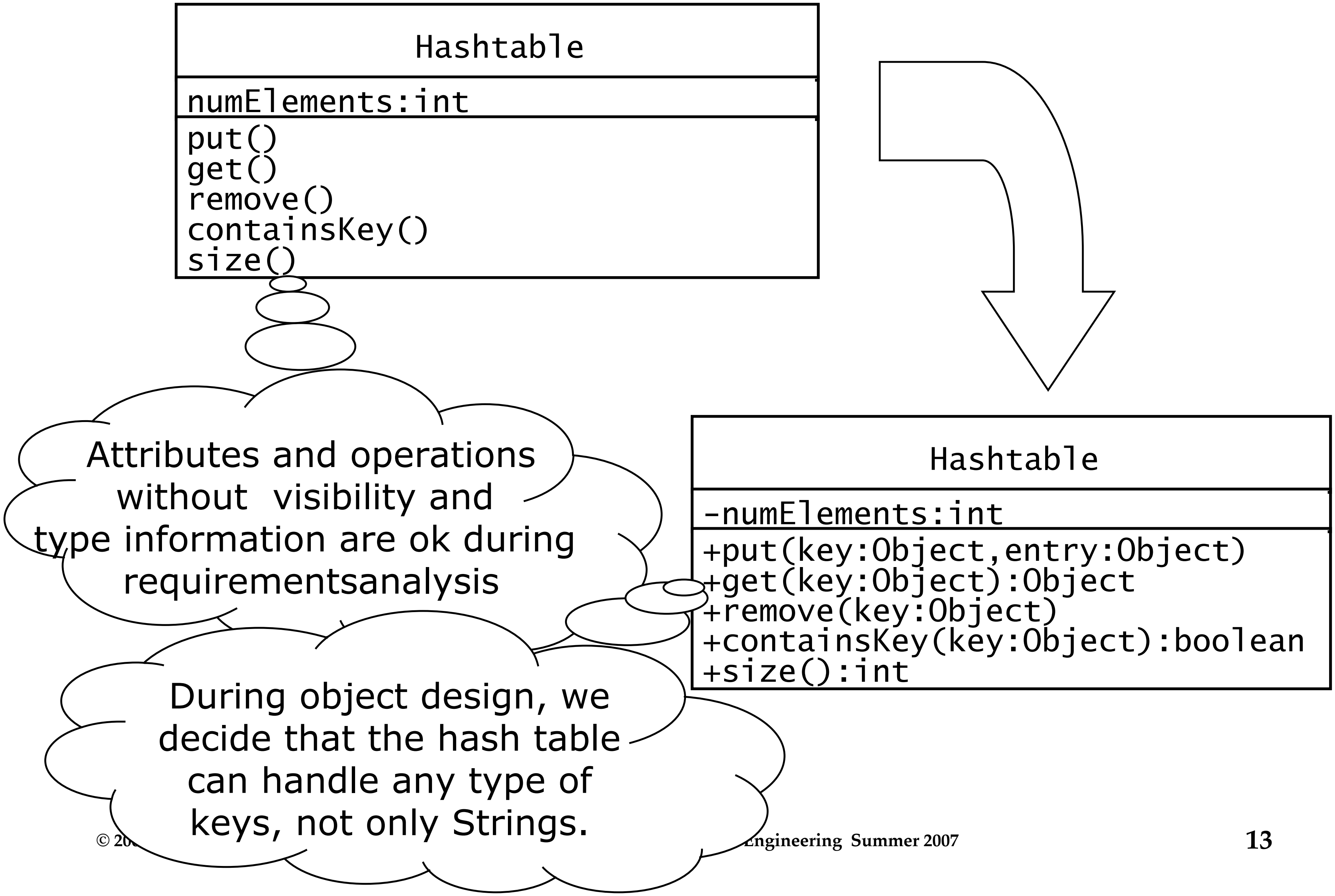
Information Hiding Heuristics

- Carefully define the public interface for classes as well as subsystems
  - For subsystems use a façade design pattern if possible
- Always apply the "Need to know" principle:
  - Only if somebody needs to access the information, make it publicly possible
    - Provide only well defined channels, so you always know the access
- The fewer details a class user has to know
  - the easier the class can be changed
  - the less likely they will be affected by any changes in the class implementation
- Trade-off: Information hiding vs. efficiency
  - Accessing a private attribute might be too slow.

Information Hiding Design Principles

- Only the operations of a class are allowed to manipulate its attributes
  - Access attributes only via operations
- Hide external objects at subsystem boundary
  - Define abstract class interfaces which mediate between the external world and the system as well as between subsystems
- Do not apply an operation to the result of another operation
  - Write a new operation that combines the two operations.

Add Type Signature Information



Outline of Today's Lecture

- Object Design Activities
- Visibilities
- Information Hiding
- Contracts

Modeling Constraints with Contracts

- Example of constraints in Arena:
  - An already registered player cannot be registered again
  - The number of players in a tournament should not be more than maxNumPlayers
  - One can only remove players that have been registered
- These constraints cannot be modeled in UML
- We model them with contracts
- Contracts can be written in OCL.

Contract

- Contract: A lawful agreement between two parties in which both parties accept obligations and on which both parties can found their rights
  - The remedy for breach of a contract is usually an award of money to the injured party
- Object-oriented contract: Describes the services that are provided by an object if certain conditions are fulfilled
  - services = “obligations”, conditions = “rights”
  - The remedy for breach of an OO-contract is the generation of an exception.

Object-Oriented Contract

- An object-oriented contract describes the services that are provided by an object. For each service, it specifically describes two things:
  - The conditions under which the service will be provided
  - A specification of the result of the service
- Examples:
  - A letter posted before 18:00 will be delivered on the next working day to any address in Germany
  - For the price of 4 Euros a letter with a maximum weight of 80 grams will be delivered anywhere in the USA within 4 hours of pickup.

Object-Oriented Contract

- An object-oriented contract describes the services that are provided by an object. For each service, it specifically describes two things:
  - The conditions under which the service will be provided
  - A specification of the result of the service that is provided.
- Examples:
  - A letter posted before 18:00 will be delivered on the next working day to any address in Germany.
  - For the price of 4 Euros a letter with a maximum weight of 80 grams will be delivered anywhere in Germany within 4 hours of pickup.

Modeling OO-Contracts

- Natural Language
- Mathematical Notation
- Models and contracts:
  - A language for the formulation of constraints with the formal strength of the mathematical notation and the easiness of natural language:

    $\Rightarrow$ UML + OCL (Object Constraint Language)
  - Uses the abstractions of the UML model
  - OCL is based on predicate calculus

Contracts and Formal Specification

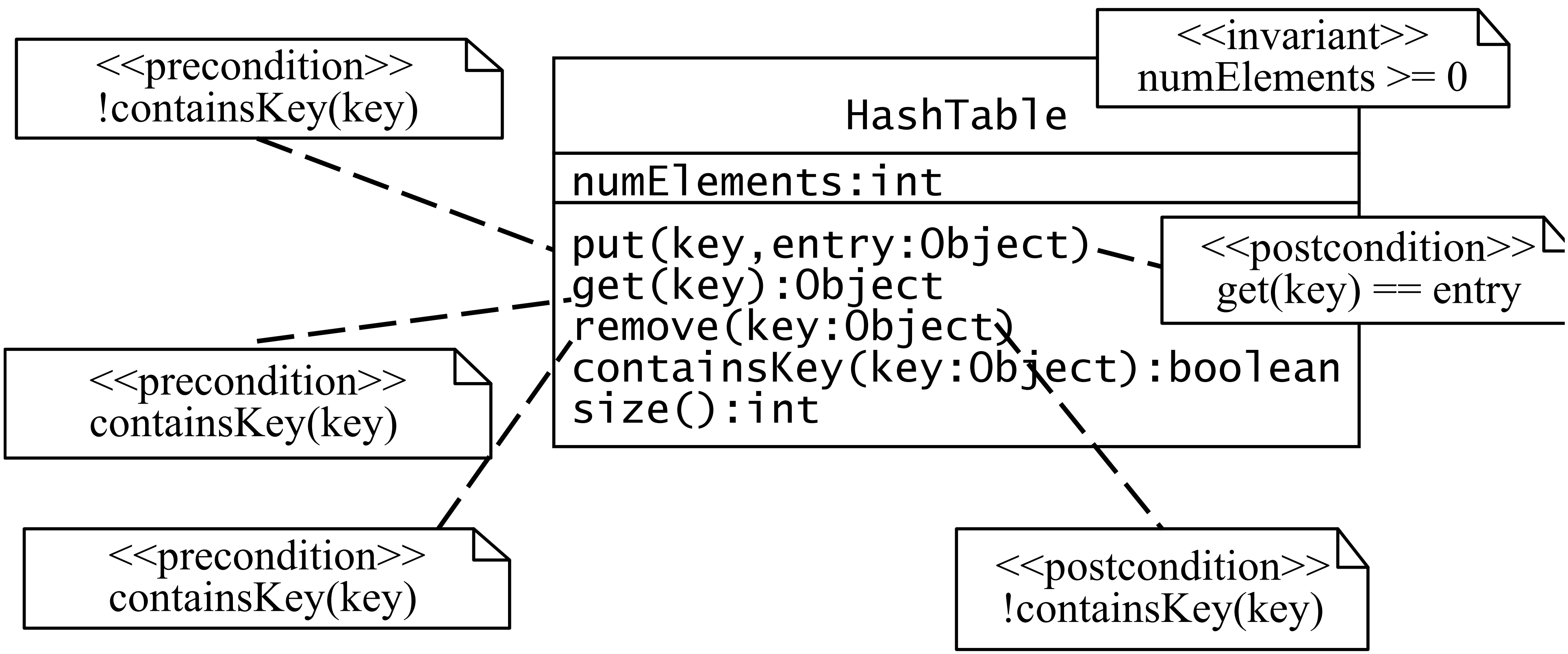
- Contracts enable the caller and the provider to share the same assumptions about the class
- A contract is an exact specification of the interface of an object
- A contract include three types of constraints:
  - Invariant:
    - A predicate that is always true for all instances of a class
  - Precondition ("rights"):
    - Must be true before an operation is invoked
  - Postcondition ("obligation"):
    - Must be true after an operation is invoked.

Formal Specification

- A contract is called a formal specification, if the invariants, rights and obligations in the contract are unambiguous.

Expressing Constraints in UML Models

- A constraint can also be depicted as a note attached to the constrained UML element by a dependency relationship.



Why not use Contracts already in Requirements Analysis?

- Many constraints represent domain level information
- Why not use them in requirements analysis?
  - Constraints increase the precision of requirements
  - Constraints can yield more questions for the end user
  - Constraints can clarify the relationships among several objects
- Constraints are sometimes used during requirements analysis, however there are trade offs

Requirements vs. Object Design Trade-offs

- Communication among stakeholders
  - Can the client understand formal constraints?
- Level of detail vs. rate of requirements change
  - Is it worth precisely specifying a concept that will change?
- Level of detail vs. elicitation effort
  - Is it worth the time interviewing the end user
  - Will these constraints be discovered during object design anyway?
- Testing constraints
  - If tests are generated early, do they require this level of precision?

To be continued in Lecture on OCL