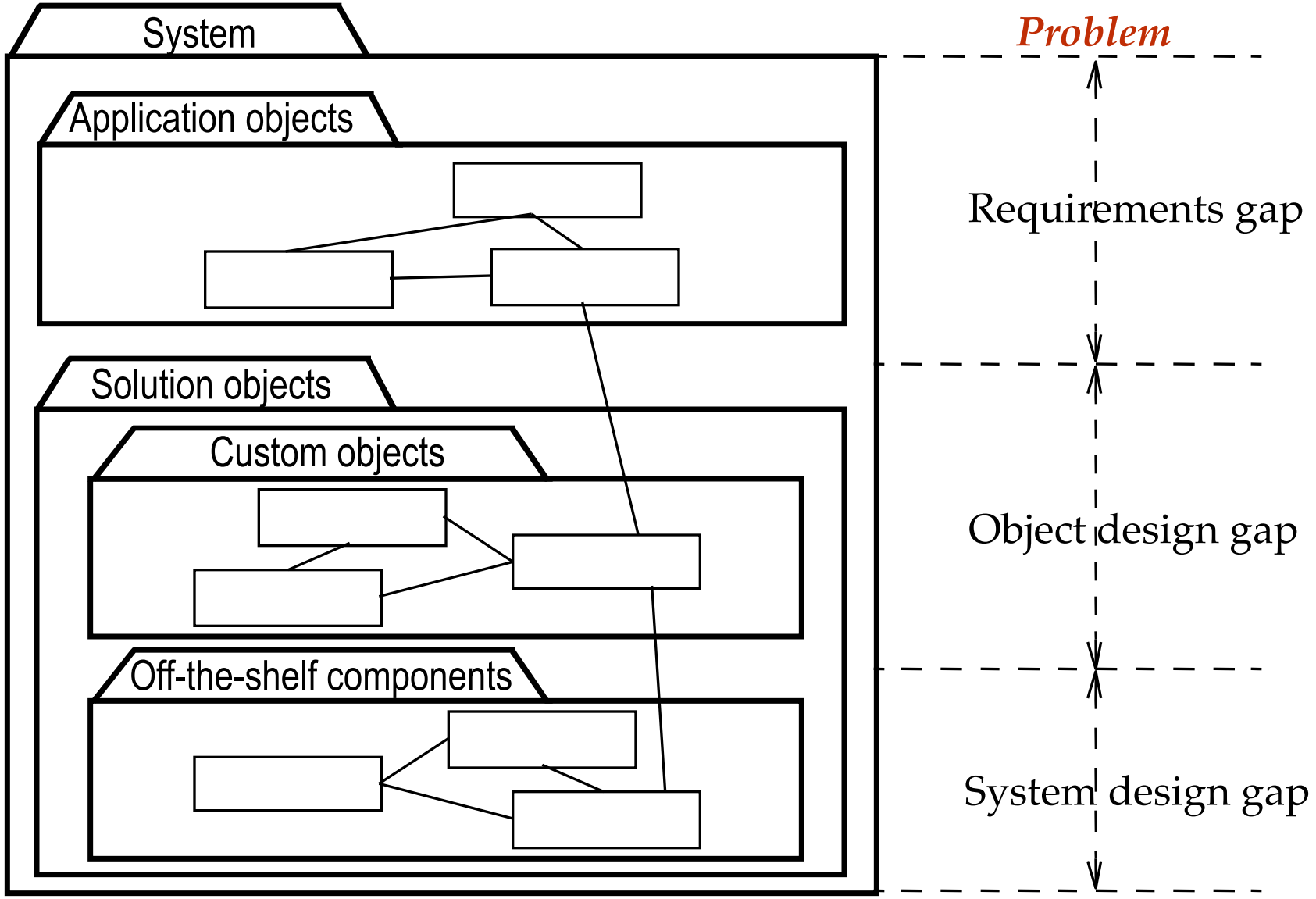
Chapter 9, Object Design: Specifying Interfaces

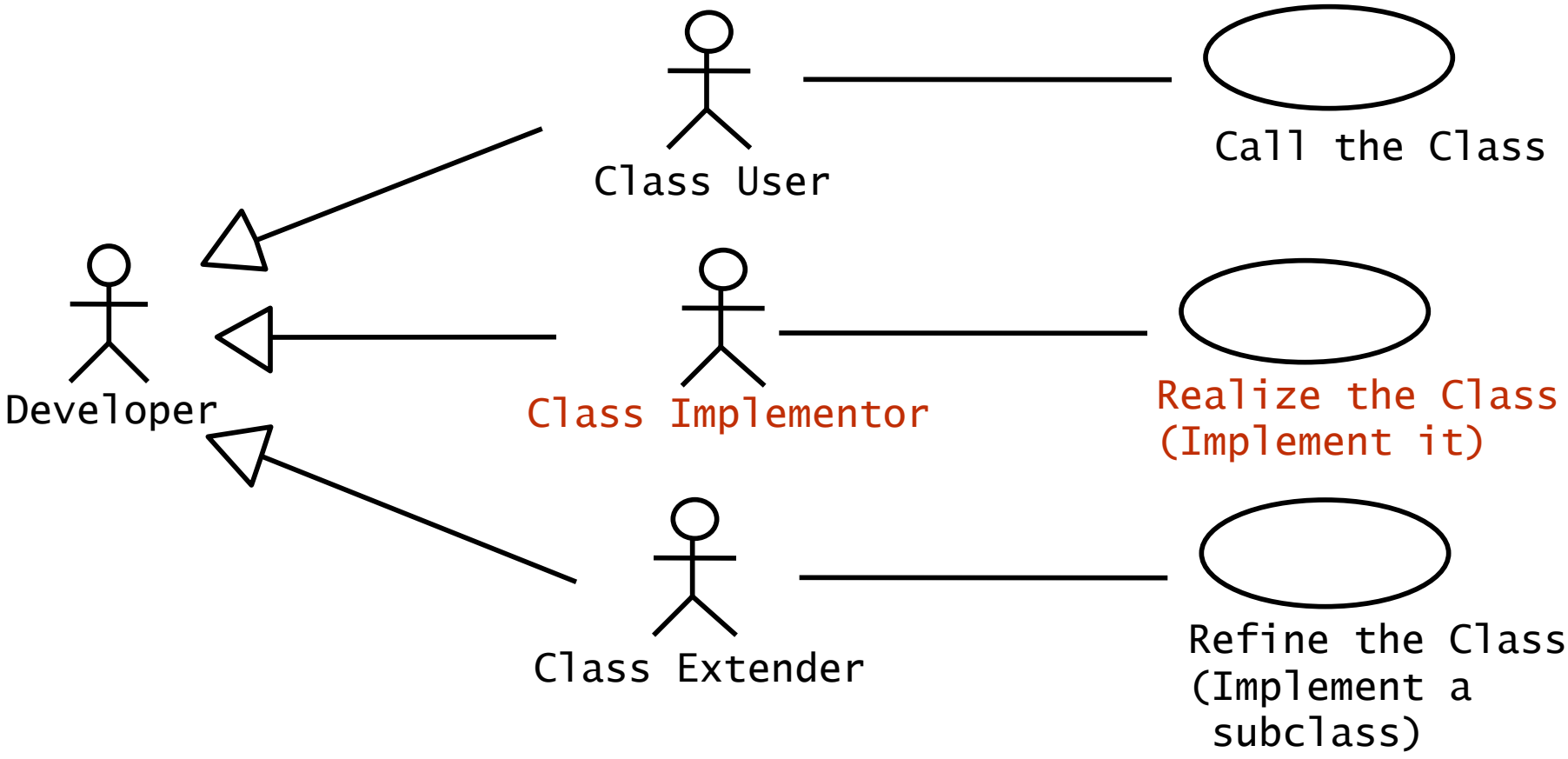
Requirements Analysis vs. Object Design

- 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.

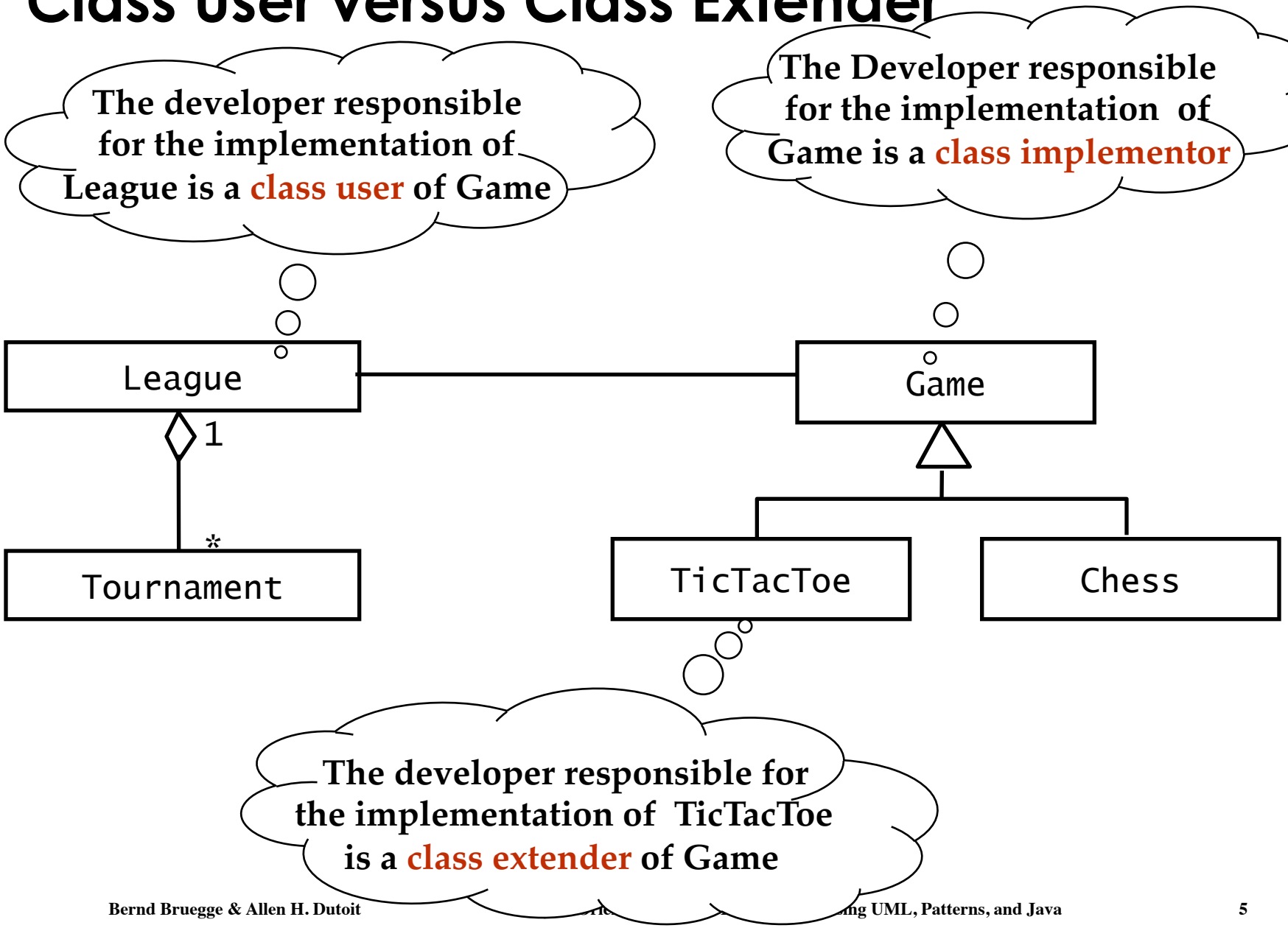
Object Design: Closing the Final Gap



Developers play 3 different Roles during Object Design of a Class



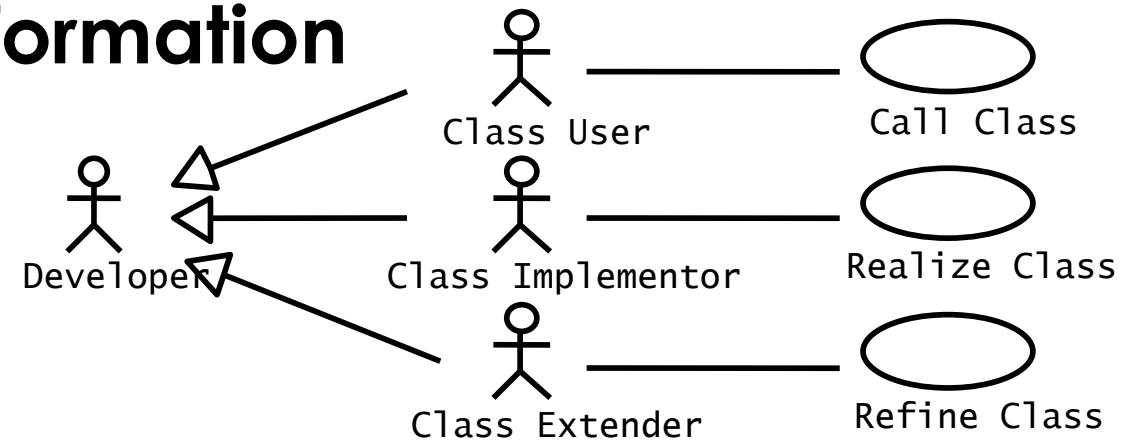
Class User versus Class Extender



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.

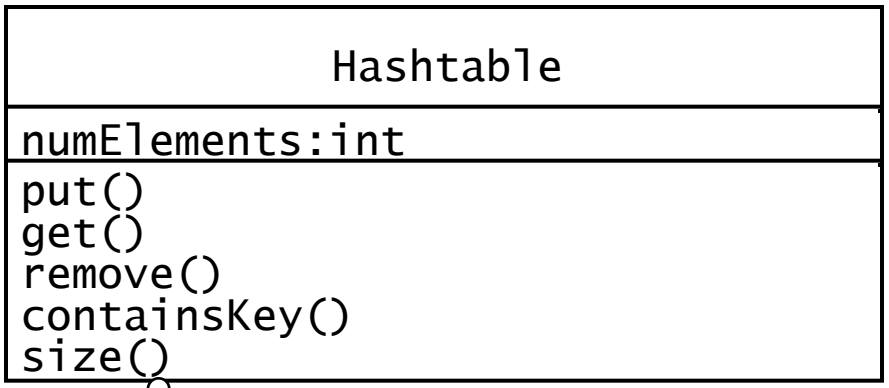
Implementation of UML Visibility in Java

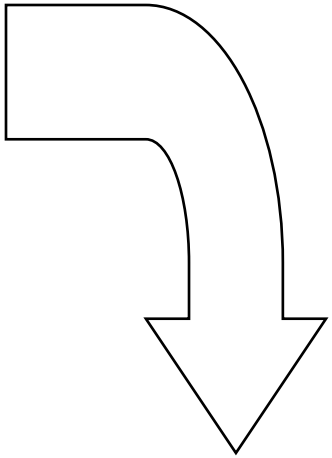

```
public class Tournament {
   private int maxNumPlayers;

   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) {…};
```

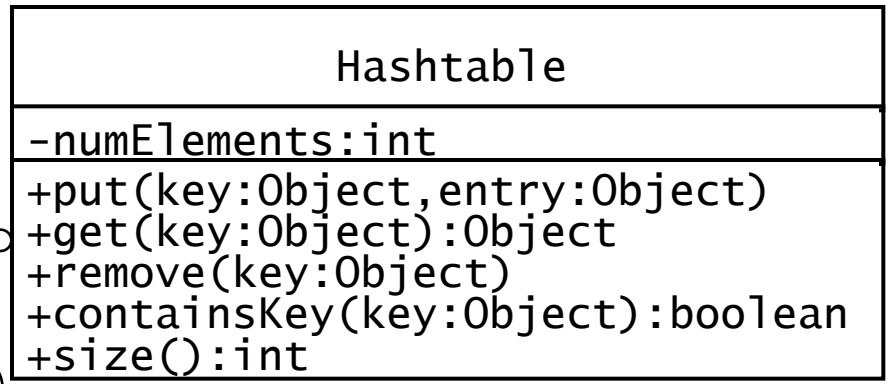
Information Hiding Heuristics

- 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.

Add Type Signature Information







Attributes and operations without visibility and type information are ok during requirementsanalysis

During object design, we decide that the hash table can handle any type of keys, not only Strings.

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
- We model them with contracts.
- These constraints can now be modeled in UML since contacts can be written in OCL, which has been made part of the UML standard.

Contracts and Formal Specification

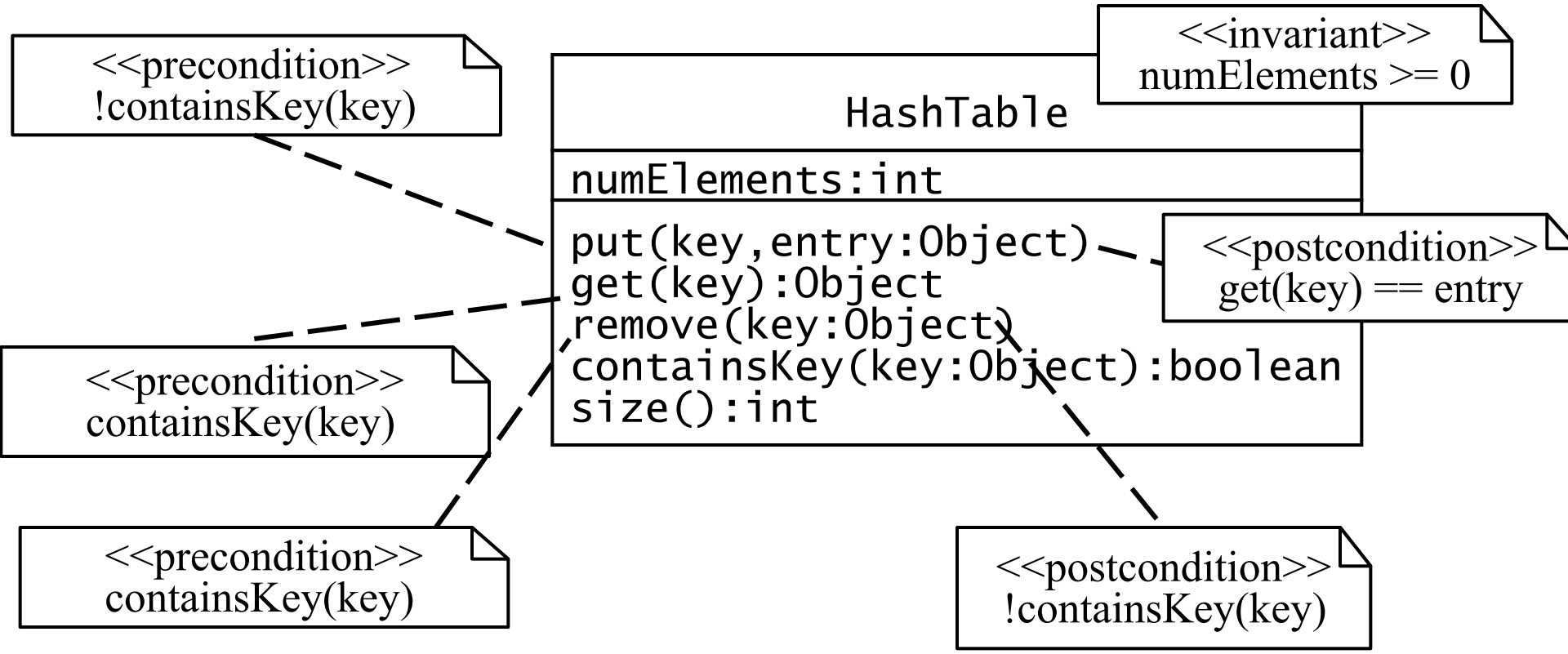
- Contracts enable the caller and the provider to share the same assumptions about the class
- A contract is an exact specification of the interface of an object
- A contract include three types of constraints:
  - Invariant:
    - A predicate that is always true for all instances of a class
  - Precondition (“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.

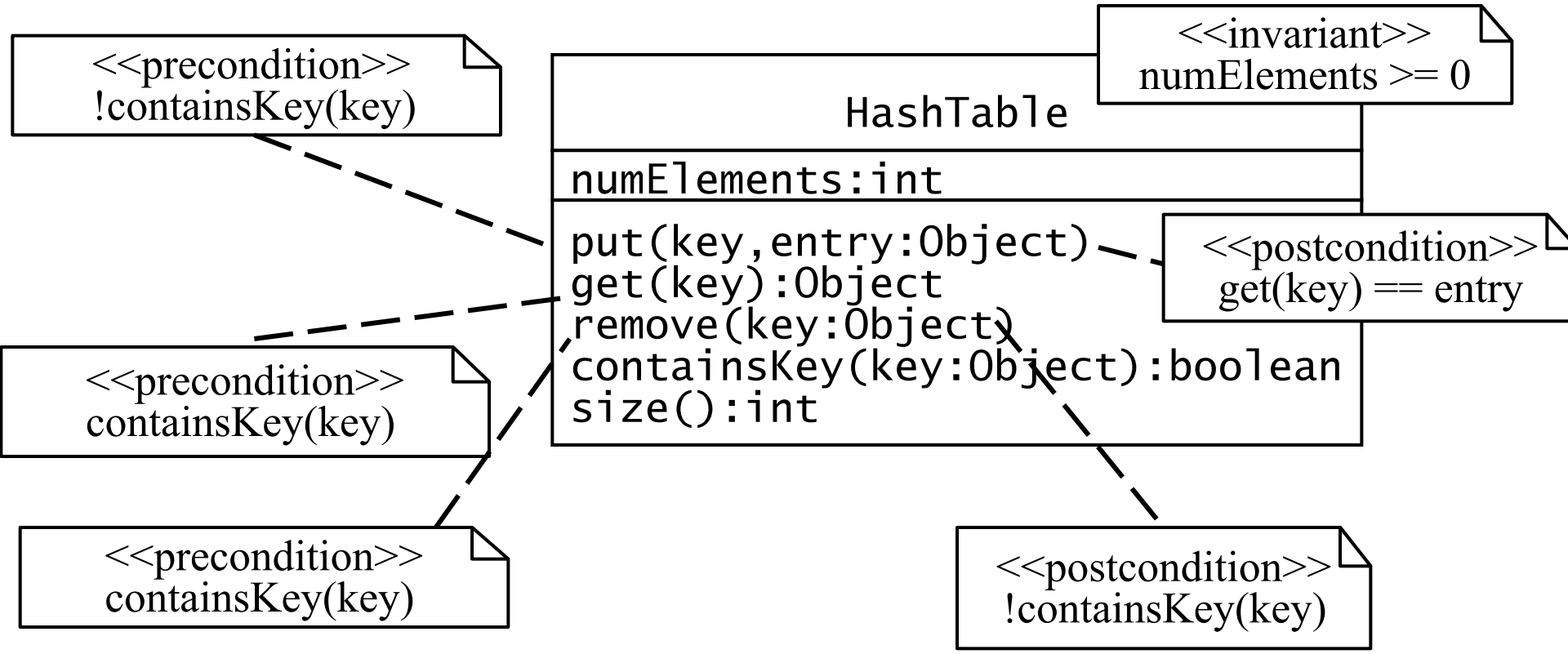
Expressing Constraints in UML Models

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



Expressing Constraints in UML Models

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



Or using OCL: Object Constraint Language

- Formal language for expressing constraints over a set of objects and their attributes
- Part of the UML standard
- Used to write constraints that cannot otherwise be expressed in a diagram
- Declarative
  - No side effects
  - No control flow
- Based on Sets and Multi Sets