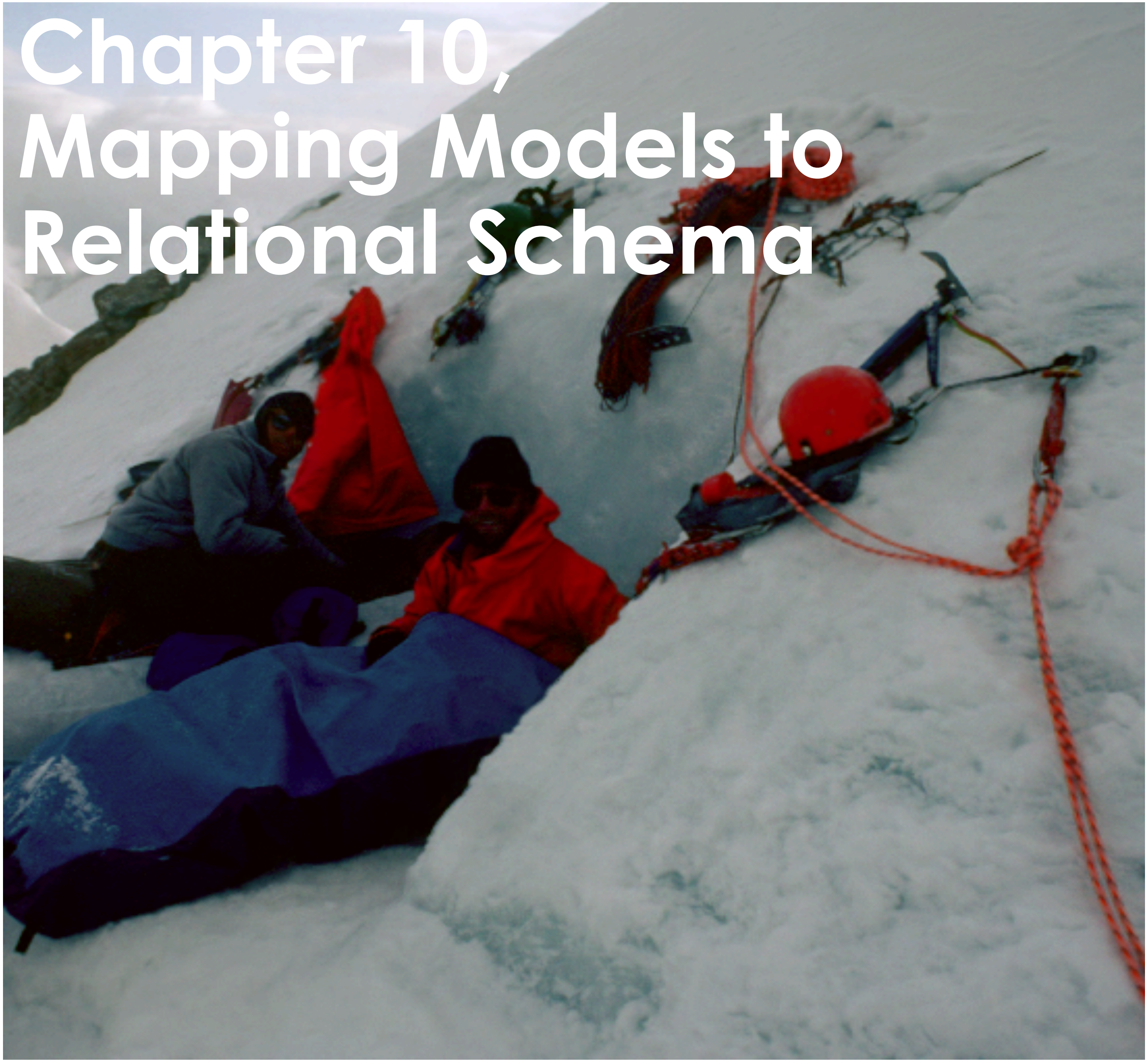
Object-Oriented Software Engineering

Using UML, Patterns, and Java

Chapter 10, Mapping Models to Relational Schema



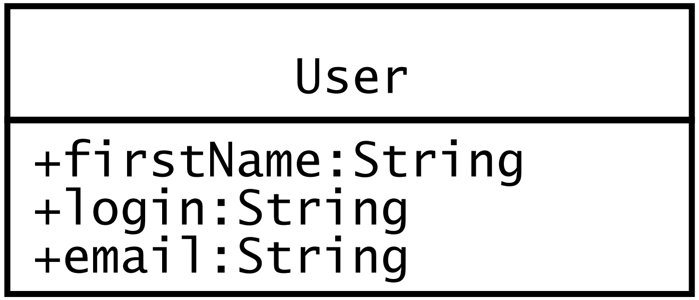
Lecture Plan

- Last lecture:
  - Operations on the object model:
    - Optimizations to address performance requirements
  - Implementation of class model components:
    - Realization of associations
    - Realization of operation contracts
- This lecture:
  - Realizing entity objects based on selected storage strategy
  - ➡ Mapping the object model to a database
  - Mapping class diagrams to tables.

Mapping an Object Model to a Database

- UML object models can be mapped to relational databases:
  - Some degradation occurs because all UML constructs must be mapped to a single relational database construct - the table
- Mapping of classes, attributes and associations
  - Each *class* is mapped to a table
  - Each class *attribute* is mapped onto a column in the table
  - An *instance* of a class represents a row in the table
  - *A many-to-many association* is mapped into its own table
  - A *one-to-many association* is implemented as buried foreign key
- Methods are not mapped.

Mapping a Class to a Table



User

+firstName:String
+login:String
+email:String

User table

id:long	firstName:text[25]	login:text[8]	email:text[32]

Primary and Foreign Keys

- Any set of attributes that could be used to uniquely identify any data record in a relational table is called a **candidate key**
- The actual candidate key that is used in the application to identify the records is called the **primary key**
  - The primary key of a table is a set of attributes whose values uniquely identify the data records in the table
- A **foreign key** is an attribute (or a set of attributes) that references the primary key of another table.

Example for Primary and Foreign Keys

User table

Primary key

firstName	login	email
“alice”	“am384”	“am384@mail.org”
“john”	“js289”	“john@mail.de”
“bob”	“bd”	“bobd@mail.ch”

Candidate key

Candidate key

League table

name	login
“tictactoeNovice”	“am384”
“tictactoeExpert”	“bd”
“chessNovice”	“js289”

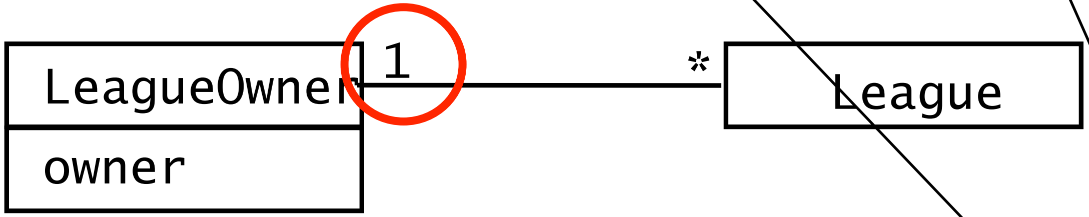
Foreign key referencing **User table**

Buried Association

- Associations with multiplicity "one" can be implemented using a foreign key

For one-to-many associations we add the foreign key to the table representing the class on the "many" end

For all other associations we can select either class at the end of the association.



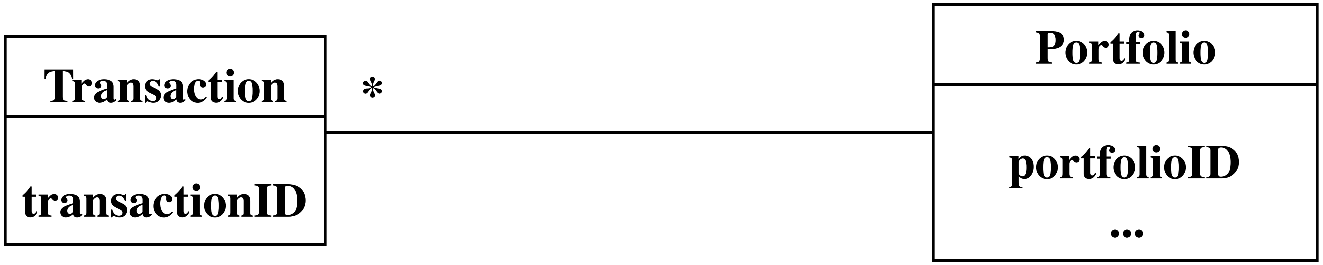
LeagueOwner table

id:long	...

League table

id:long	...	owner:long

Another Example for Buried Association



Transaction Table

transactionID	portfolioID

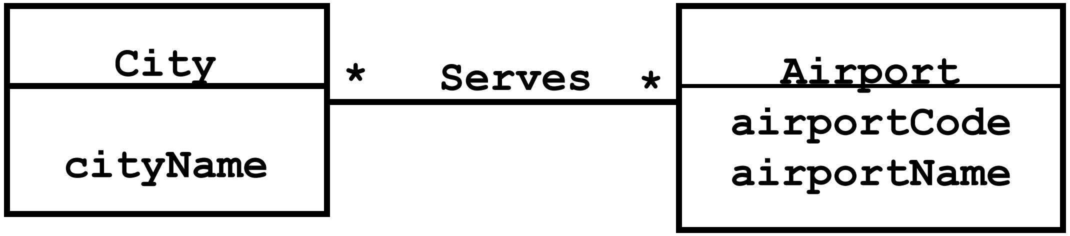
Foreign Key

Portfolio Table

portfolioID	...

Mapping Many-To-Many Associations

In this case we need a separate table for the association



Separate table for the association “Serves”

Primary Key

City Table

cityName
Houston
Albany
Munich
Hamburg

Airport Table

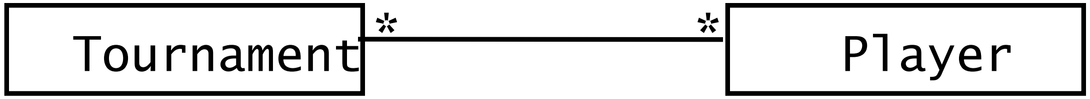
airportCode	airportName
IAH	Intercontinental
HOU	Hobby
ALB	Albany County
MUC	Munich Airport
HAM	Hamburg Airport

Serves Table

cityName	**airportCode**
Houston	IAH
Houston	HOU
Albany	ALB
Munich	MUC
Hamburg	HAM

Another Many-to-Many Association Mapping

We need the Tournament/Player association as a separate table



Tournament table

id	name	...
23	novice	
24	expert	

TournamentPlayerAssociation table

tournament	player
23	56
23	79

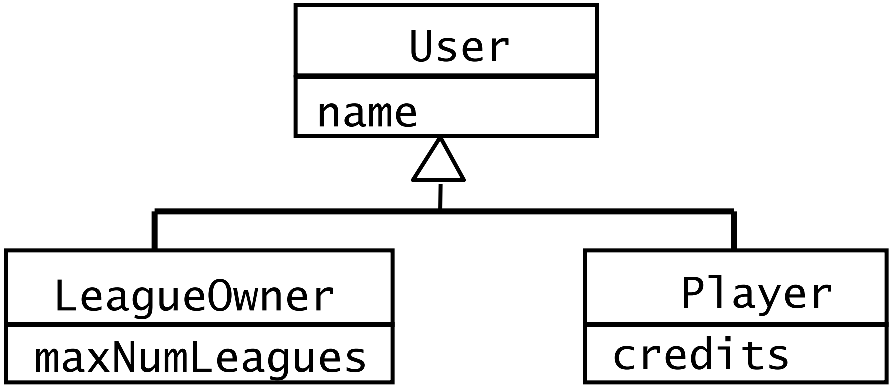
Player table

id	name	...
56	alice	
79	john	

Realizing Inheritance

- Relational databases do not support inheritance
- Two possibilities to map an inheritance association to a database schema
  - ➡ With a separate table (”vertical mapping”)
    - The attributes of the superclass and the subclasses are mapped to different tables
  - By duplicating columns (”horizontal mapping”)
    - There is no table for the superclass
    - Each subclass is mapped to a table containing the attributes of the subclass and the attributes of the superclass

Realizing inheritance with a separate table (Vertical mapping)



User table

id	name	...	role
56	zoe		LeagueOwner
79	john		Player

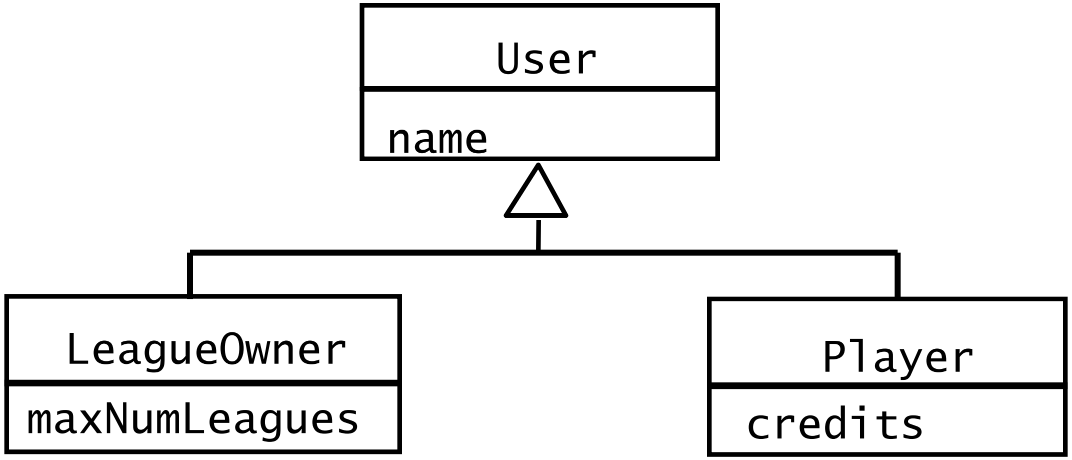
LeagueOwner table

id	maxNumLeagues	...
56	12	

Player table

id	credits	...
79	126	

Realizing inheritance by duplicating columns (Horizontal Mapping)



User

name

LeagueOwner

maxNumLeagues

Player

credits

LeagueOwner table

id	name	maxNumLeagues	...
56	zoe	12	

Player table

id	name	credits	...
79	john	126	

Comparison: Separate Tables vs Duplicated Columns

- The trade-off is between modifiability and response time
  - How likely is a change of the superclass?
  - What are the performance requirements for queries?
- Separate table mapping (Vertical mapping)
  - ☺We can add attributes to the superclass easily by adding a column to the superclass table
  - ☹Searching for the attributes of an object requires a join operation.
- Duplicated columns (Horizontal Mapping)
  - ☹Modifying the database schema is more complex and error-prone
  - ☺Individual objects are not fragmented across a number of tables, resulting in faster queries

Summary

- Four mapping concepts:
  - Model transformation improves the compliance of the object design model with a design goal
  - Forward engineering improves the consistency of the code with respect to the object design model
  - Refactoring improves code readability/modifiability
  - Reverse engineering discovers the design from the code.
- Model transformations and forward engineering techniques:
  - Optimizing the class model
  - Mapping associations to collections
  - Mapping contracts to exceptions
  - Mapping class model to storage schemas.

Backup and Example Slides