Mapping Models to Code

Modified from Bernd Bruegge and Allen H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition, Pearson, 2013.

Mapping Activities: Optimizing the Object Design Model

- Optimizing access paths
  - Repeated association traversals
    - should identify operations that are invoked often (look at a sequence diagram)
    - frequent operations should not require many traversals, but should have a direct connection between the querying object and the queried object
    - if the direct connection is missing, add and association between those objects

Mapping Activities: Optimizing the Object Design Model (cont.)

- "Many" associations
  - try to decrease the search time by reducing the "many" to "one"
  - if it is not possible to reduce, try to order or index the objects on the "many" side to decrease access time
- Misplaced attributes
  - many classes identified during the analysis may have no interesting behavior
  - if they are only involved in set() and get(), try to fold these attributes into the calling class

Mapping Activities: Optimizing the Object Design Model (cont.)

- Collapsing objects: Turning objects into attributes
  - after the object model is restructured and optimized, some of its classes may have few attributes or behaviors left
  - such class can be collapsed into an attribute

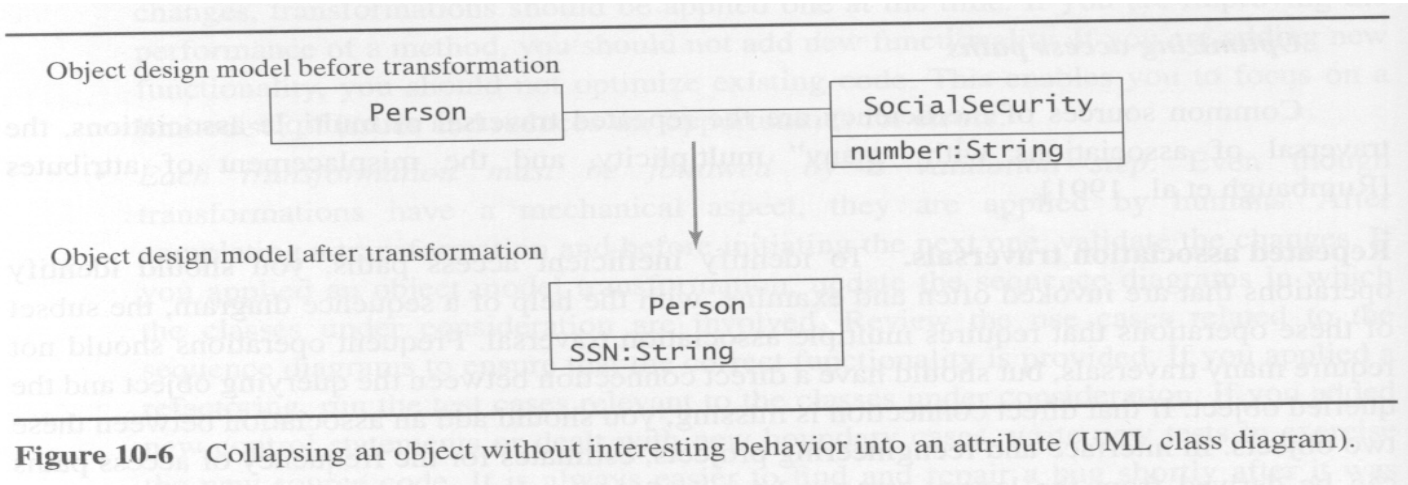


Figure 10-6 Collapsing an object without interesting behavior into an attribute (UML class diagram).

Mapping Activities: Optimizing the Object Design Model (cont.)

- Delaying expensive computations
  - often, specific objects are expensive to create
  - however, their creation can often be delayed until their actual content is needed

Figure 10-7: Delaying Expensive Computations to Transform the Object Design Model using a Proxy Design Pattern

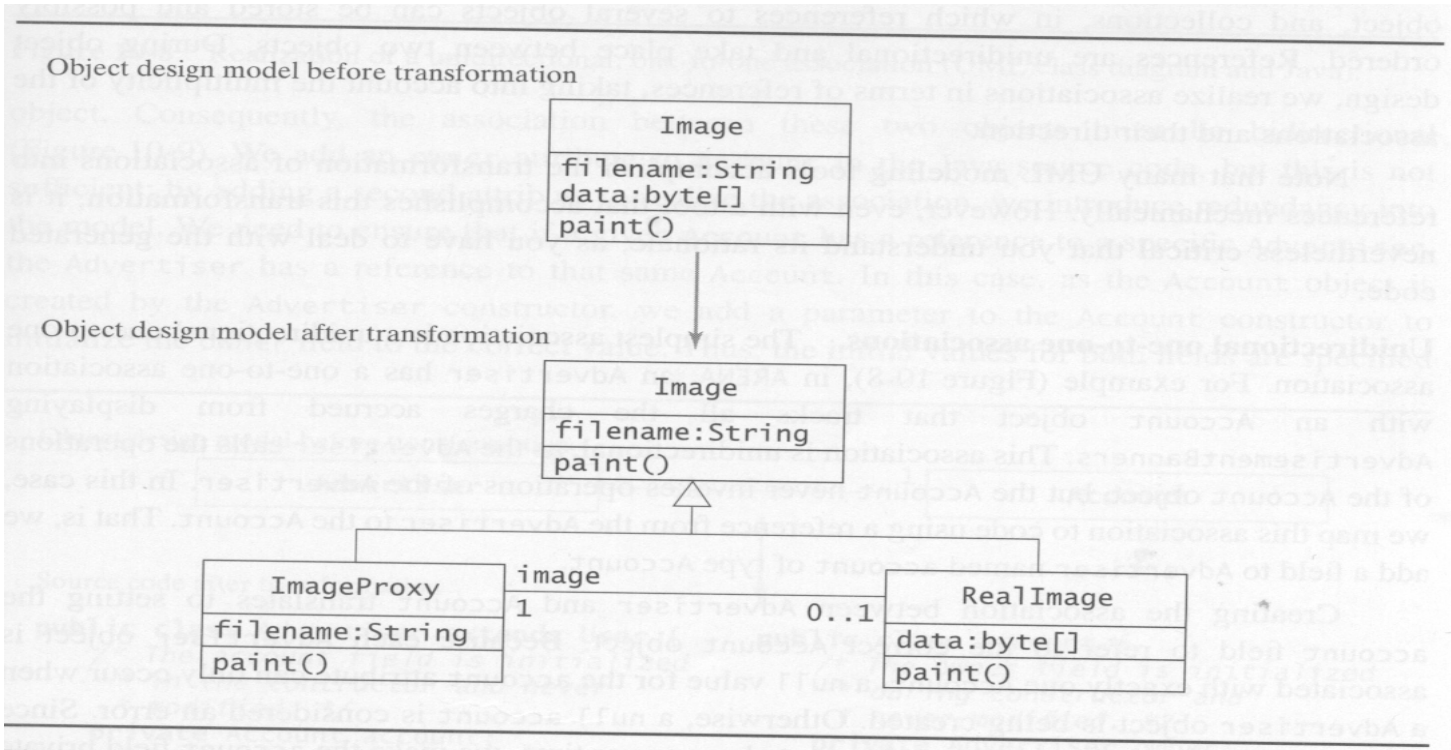


Figure 10-7 Delaying expensive computations to transform the object design model using a Proxy design pattern (UML class diagram).

Mapping Activities: Optimizing the Object Design Model (cont.)

- Caching the result of expensive computations
  - some methods are called many times, but their results are based on values that do not change or change only infrequently
  - reducing number of computations required by these methods improve overall response time
  - the result of the computation should be cached as a private attribute

Mapping Activities: Mapping Associations to Collections

- Associations are UML concepts that denote collections of bidirectional links between two or more objects
- Object-oriented programming languages, however, do not provide the concept of association
- Instead, they provide references, in which one object stores a handle to another object
- During object design, we realize associations in terms of references, taking into account the multiplicity of the associations and their direction

Mapping Activities: Mapping Associations to Collections

- Unidirectional one-to-one associations
  - in ARENA, an Advertiser has a one-to-one association with an Account object
  - the association is unidirectional
    - Advertiser calls the operations of the Account object, but
    - Account never invokes operation of the Advertiser

Figure 10-8: Realization of a Unidirectional, one-to-one Association

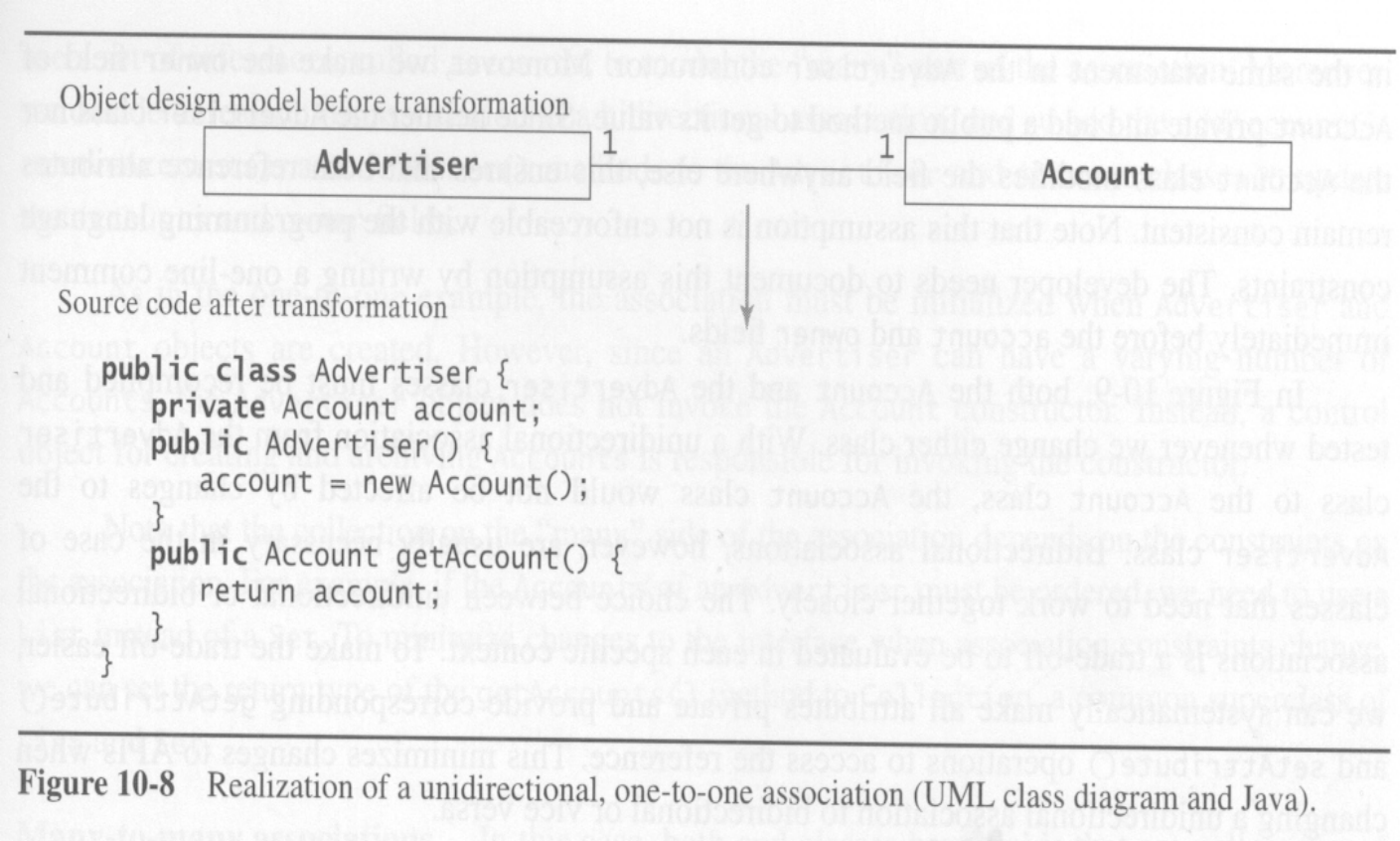


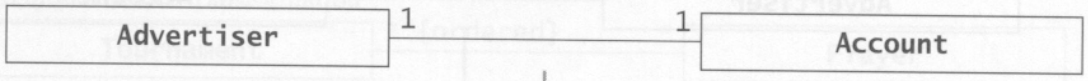
Figure 10-8 Realization of a unidirectional, one-to-one association (UML class diagram and Java).

Mapping Activities: Mapping Associations to Collections

- Bidirectional one-to-one associations
- One-to-many associations
  - realize the “many” part using a collection of references
- Many-to-many associations
  - both end classes have fields that are collections of references and operations to keep these collections consistent

Figure 10-9: Realization of a Bidirectional, One-to-One Association

Object design model before transformation



Source code after transformation

```
public class Advertiser extends User {
   /* The account field is initialized
    * in the contructor and never
    * modified. */
   private Account account;

   public Advertiser() {
      account = new Account (this);
   }
   public Account getAccount() {
      return account;
   }
}
```

```
public class Account {
   /* The owner field is initialized
    * during constructor and
    * never modified. */
   private Advertiser owner;

   public Account(owner:Advertiser) {
      this.owner = owner;
   }
   public Advertiser getOwner() {
      return owner;
   }
}
```

Figure 10-9 Realization of a bidirectional one-to-one association (UML class diagram and Java excerpts).

Figure 10-10: Realization of a Bidirectional, One-to-Many Association

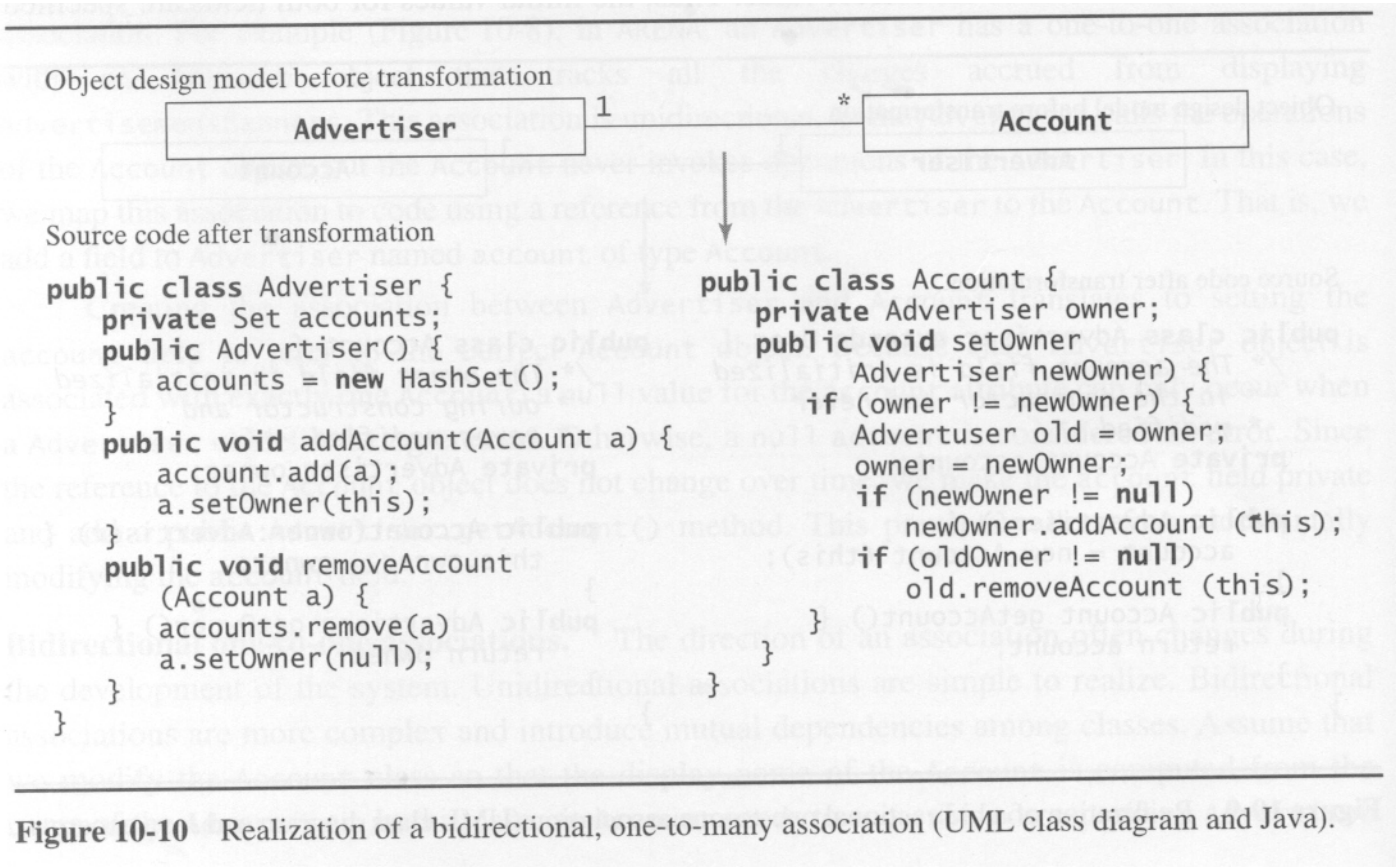
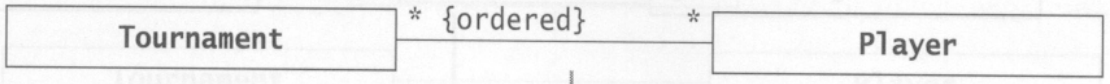


Figure 10-10 Realization of a bidirectional, one-to-many association (UML class diagram and Java).

Figure 10-11: Realization of a Bidirectional, Many-to-Many Association

Object design model before transformation



Source code after transformation

```
public class Tournament {
   private List players;
   public Tournament() {
      players = new ArrayList();
   }
   public void addPlayer(Player p) {
      if (!players.contains(p) {
         players.add(p);
         p.addTournament(this);
      }
   }
}
```

```
public class Player {
   private List tournaments;
   public Player() {
      tournaments = new ArrayList();
   }
   public void addTournament
      (Tournament t) {
      if (!tournaments.contains(t)) {
         tounaments.add(t);
         t.addPlayer(this);
      }
   }
}
```

Figure 10-11 Realization of a bidirectional, many-to-many association (UML class diagram and Java).

Mapping Activities: Mapping Associations to Collections

- Qualified associations
  - are used to reduce the multiplicity of one “many” side in a one-to-many or a many-to-many association
  - the qualifier of the association is an attribute of the class on the “many” side of the association
- Association classes
  - we use an association class to hold the attributes and operations of an association
  - to realize such an association
    - transform the association class into a separate object and a number of binary association

Figure 10-12: Realization of a Bidirectional Qualified Association

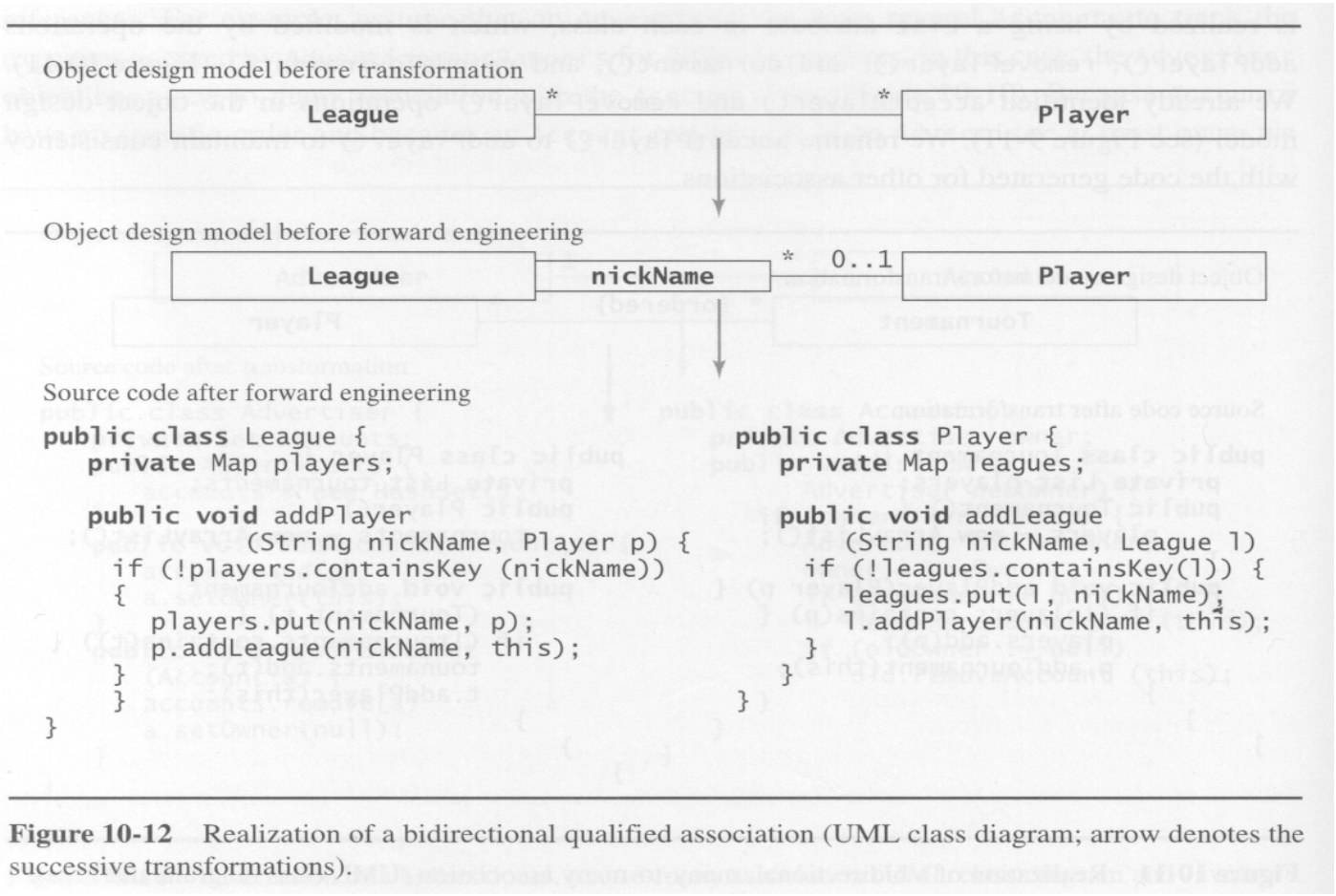


Figure 10-12 Realization of a bidirectional qualified association (UML class diagram; arrow denotes the successive transformations).

Figure 10-13: Transformation of an Association Class into An Object and Two Binary Associations

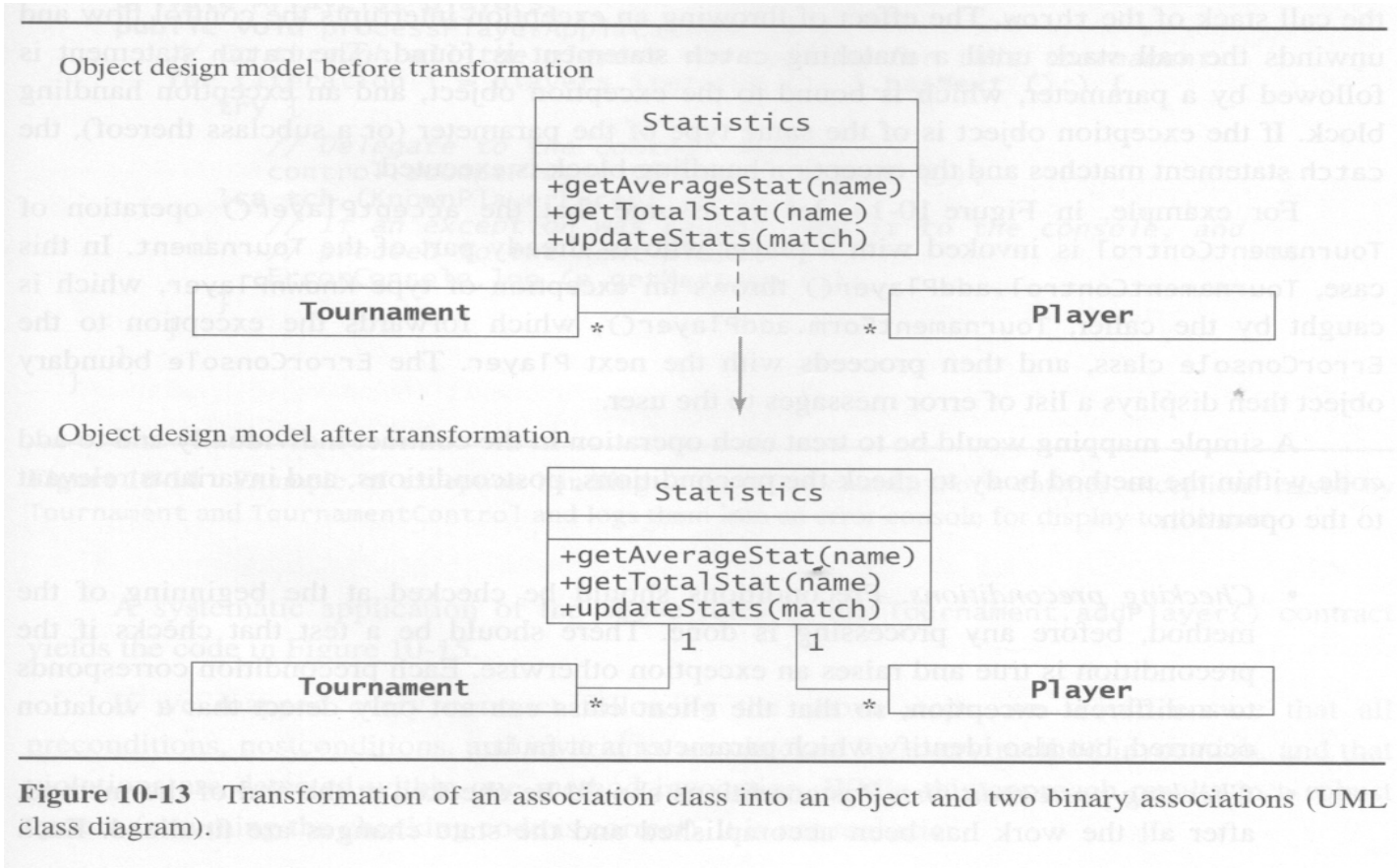


Figure 10-13 Transformation of an association class into an object and two binary associations (UML class diagram).

สรุป 1 – MaptoCode

- Optimizing the existing design model ก่อนโดย
  - Repeated association traversals

    ดูจาก Seq. D. ถ้าเราพบว่า methods ใดถูกเรียกใช้บ่อยมาก แล้วผู้เรียกใช้ต้องเรียกผ่านหลายทอด ในลักษณะ long traversal แล้ว ก็ให้พิจารณาเชื่อมต่อตรงโดยการเพิ่ม direct association ให้เลยก็ได้
  - ลด “Many” multiplicity in the association

    เพราะการมี many multiplicity ใน association ตอน coding จะเสียพื้นที่ในการเก็บ OID link และอาจจะเสียเวลาในการค้น

สรุป 2

- Optimizing the existing design model ก่อนโดย
  - ใช้เทคนิค bad smell/design patterns มาช่วยในการ optimize design เช่น
    - การที่มี lazy class คือมีแต่ attributes แต่ไม่ค่อยมี methods ที่สำคัญ เราควร collapse class
    - การ delay expensive computation โดยใช้ proxy patterns

สรุป 3

- เมื่อพบว่า design model ผ่านการ optimizing แล้ว
- เริ่มทำการ map to code
  - Class structure (ชื่อ Class, attribute names, method names และ visibility mode ของ attributes และ methods) การตั้งชื่อที่เหมาะสม เข้าใจง่าย และชื่อควรตรงกัน
  - ระหว่าง Class คู่หนึ่งใด ที่มี association
    - Unidirectional/Bidirectional association
    - Multiplicity in association (1-to-1, 1-to-many, many-to-many)

สรุป 4

- เมื่อพบว่า design model ผ่านการ optimizing แล้ว
- เริ่มทำการ map to code
  - Class structure (ชื่อ Class, attribute names, method names และ visibility mode ของ attributes และ methods) การตั้งชื่อที่เหมาะสม เข้าใจง่าย และชื่อควรตรงกัน
  - ระหว่าง Class คู่หนึ่งใด ที่มี association
    - Unidirectional/Bidirectional association
    - Multiplicity in association (1-to-1, 1-to-many, many-to-many)
    - กรณี many เราใช้ array list เก็บ OID links
    - กรณี many-to-many เราใช้ qualified association มาแก้ ช่วยให้ด้าน many กลายเป็น one ได้

- กรณีไม่ได้ Dev. ด้วย OOP จะตรวจสอบได้อย่างไรบ้าง
- เราตรวจจาก UML profile for RestAPI design ได้ไหม

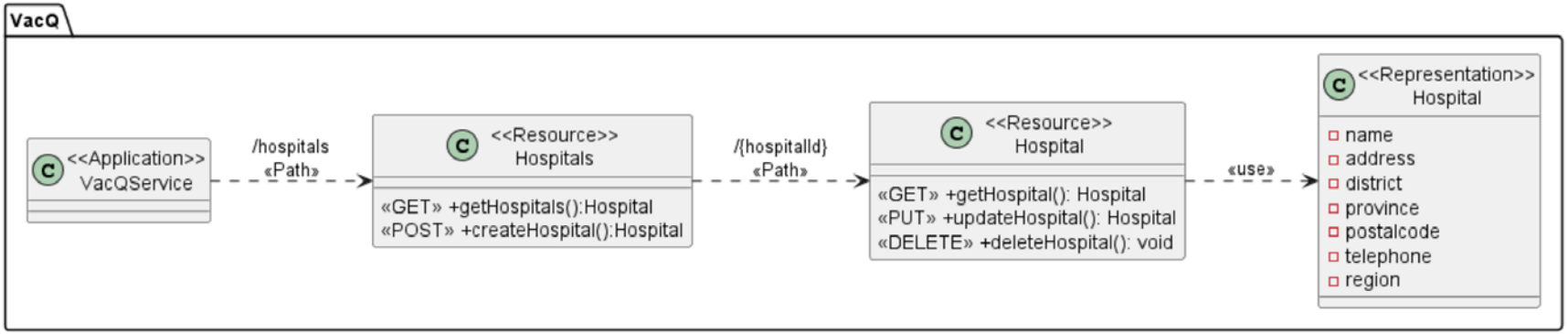
ต่อไปนี้เป็น

ตัวอย่าง Backend API ที่พัฒนา

ด้วย Node.JS และ Express framework

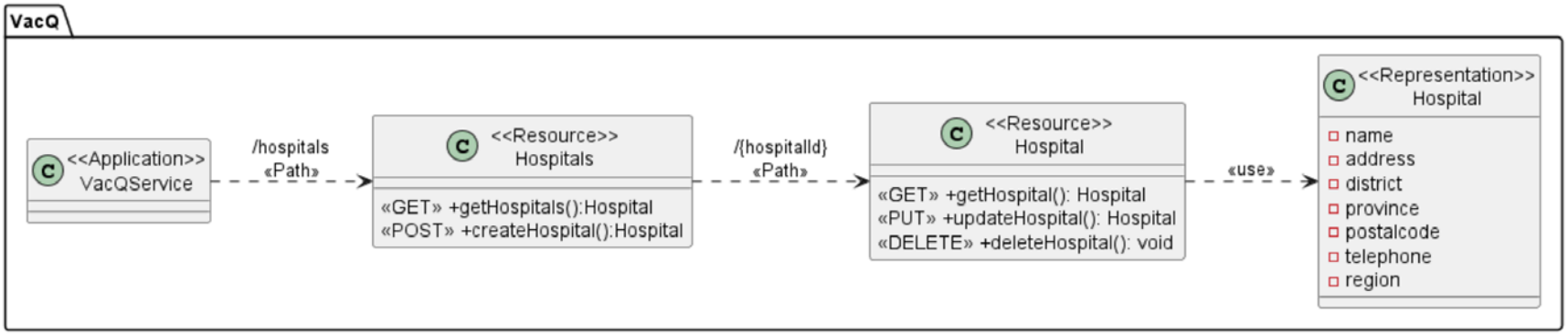
ที่เป็นส่วนหนึ่งของระบบ Hospital

Mapping UML for REST API to Node.JS with Express Framework



สิ่งที่สำคัญในการตรวจสอบ

- <<Path>> มีกี่ path? คืออะไรบ้าง
- <<Resource>> มีกี่ resource คืออะไรบ้าง
- HTTP methods : <<GET>> <<POST>> มีกี่ method คืออะไรบ้าง
- <<Representation>> มีกี่ representation คืออะไรบ้าง



2 Paths

- /hospitals
- /hospitals/001

2 Resources

- Hospitals
- Hospital

1 Representation

- Hospital

Resources “Hospitals”

- GET to read hospital list
- POST to create a hospital

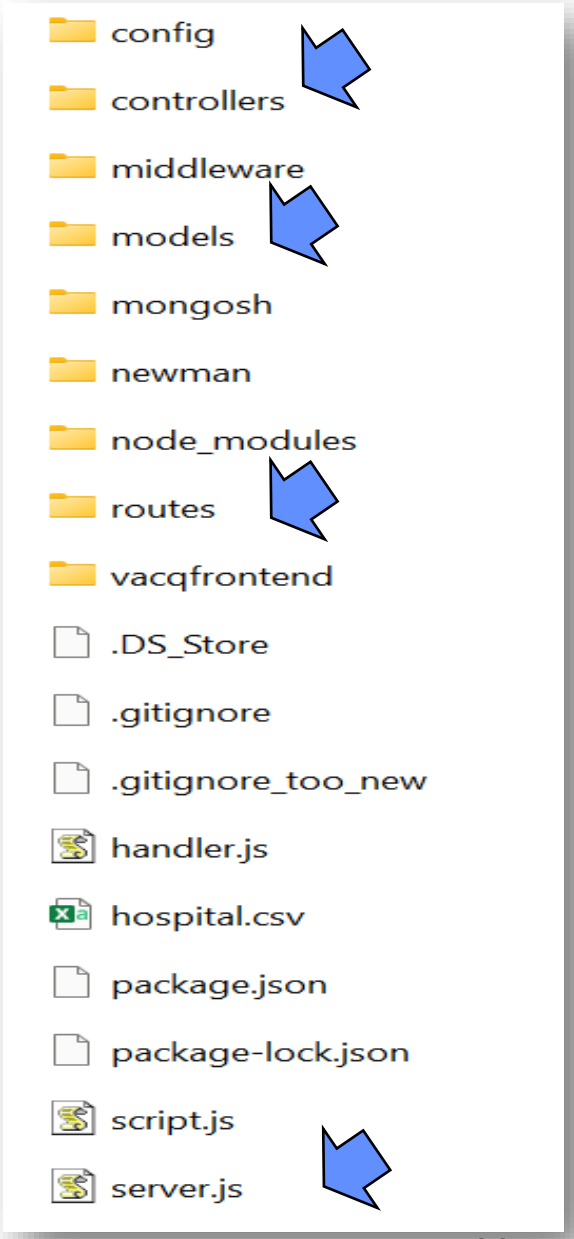
Resources “Hospital”

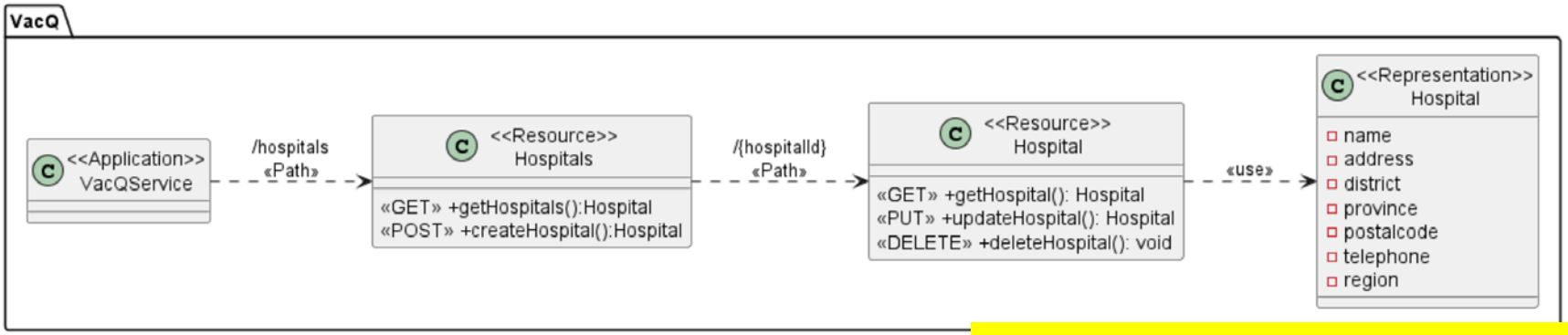
- GET to read a hospital
- PUT to update a hospital
- DELETE to remove a hospital

Node.JS with Express Framework

- เริ่มที่ server.js
- ตรวจดู path ใน /routes
- ตรวจดู resource ใน /routes
- ตรวจดู HTTP methods ใน /controllers
- ตรวจดู representation ใน /models

หรือ อาจจะแตกต่างกันถ้าใช้ framework ต่างกัน





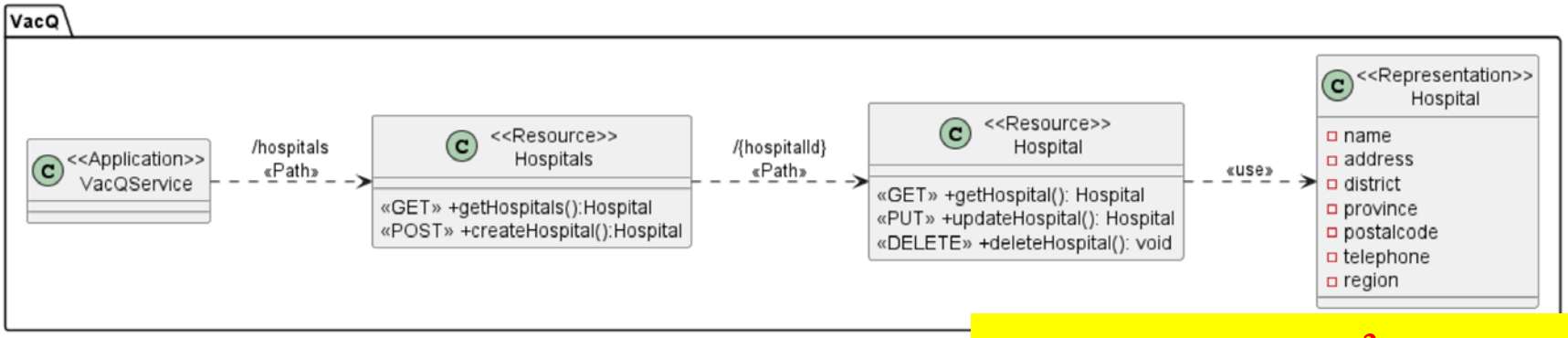
ตรวจดู paths และ resources ใน /routes

/server.js

```
//Mount routers
app.use('/api/v1/hospitals',hospitals);
app.use('/api/v1/auth',auth);
app.use('/api/v1/appointments', appointments);
```

/routes/hospitals.js

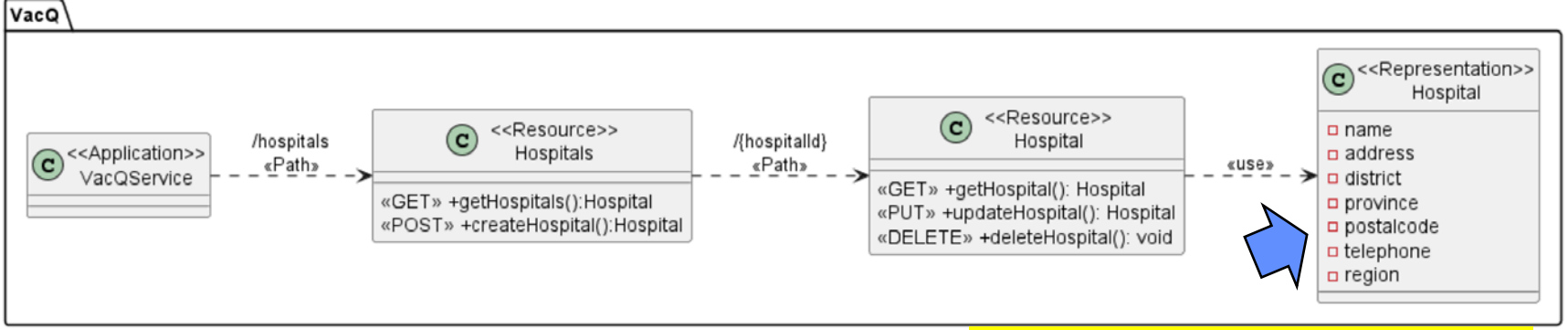
```
//Re-route into other resource routers
router.use('/:hospitalId/appointments/',appointmentRouter);

router.route('/').get(getHospitals).post(protect, authorize('admin'), createHospital);
router.route('/:id').get(getHospital).put(protect, authorize('admin'), updateHospital).de

module.exports=router;
```



ตรวจดู HTTP methods ใน /controllers

/controllers/hospitals.js

```
1   const Hospital = require('../models/Hospital.js');
2
3   //@desc     Get all hospitals
4   //@route    GET /api/v1/hospitals
5   //@access   Public
6   exports.getHospitals= async (req,res,next)=>{
7       try{
8           let query;
9
10          //Copy req.query
11          const reqQuery= {...req.query};
12
13          //Fields to exclude
14          const removeFields=['select','sort','page','limit'];
15
16          //Loop over remove fields and delete them from reqQuery
17          removeFields.forEach(param=>delete reqQuery[param]);
18          console.log(reqQuery);
19
20          //Create query string
21          let queryStr=JSON.stringify(req.query);
22          queryStr=queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);
23
```



ตรวจดู representation ใน /models

```
const mongoose = require('mongoose');
const HospitalSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true,'Please add a name'],
        unique: true,
        trim:true,
        maxlength:[50,'Name can not be more than 50 characters'],
    },
    address:{
        type: String,
        required: [true,'Please add an address'],
    },
    district:{
        type: String,
        required: [true,'Please add a district']
    },
    province:{
        type: String,
        required: [true,'Please add a province']
    },
    postalcode:{
        type: String,
        required: [true,'Please add a postalcode'],
        maxlength:[5,'Postal Code can not be more than 5 digits']
    },
```

/models/hospitals.js

Q&A