# Classes: Relationships Among Objects

Atul Prakash

Background readings: Chapters 8-11 (Downey)

# Real-World

- Relationships:
  - Parent-child relationships among members of a species
  - Friends relationship among users on Facebook
  - Students who part of the same team
  - City-city relationship for a flight network

In the above cases, two objects of the same class have a relationship with each other

# Multiple Classes

- A program will often use multiple classes
- E.g.: for handling data in IMDB, classes could be
  - Movie, Actor, User, Review, Ratings, ...
- In Java, each class will be in its own .java file:
  - Movie.java, Actor.java, User.java, Review.java, Ratings.java, etc.

# Relationships among objects from multiple classes

- Classes: Student, Course, Professor, Classroom
  - Student <-> Course
  - Professor <-> Course
  - Course <-> Classroom

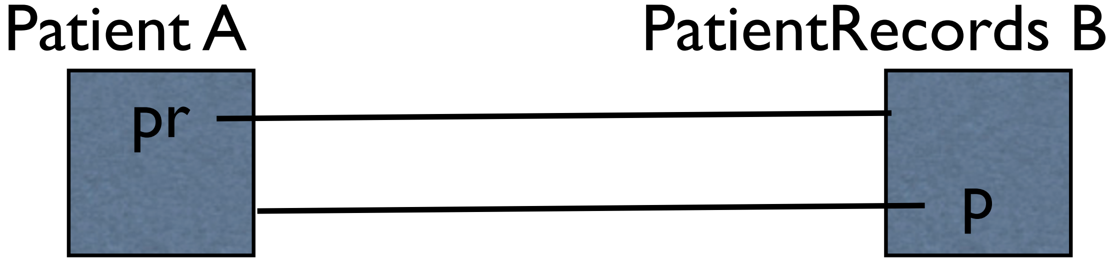
- Classes: Movie, Actor, User, Review
  - Movie <-> Actor
  - User <-> Review
  - Movie <-> Review

# Types of Relationships

- One-to-one: Patient <-> Patient Record
- One-to-many or many-to-one: Person (Mom) <-> Person (child)
  - A mom can have 1 or more children
- Many-to-many: Student <-> Course
  - A student can take many courses
  - A course can be taken by many students

# One-to-one relationships

- One-to-one relation among objects A and B
- One way to represent it both ways:
  - A contains a reference to B
  - B contains a reference to A



# Example

- Patient and PatientRecord

```
public class Patient {
    private String name;
    private String socialsecuritnumber;

    private PatientRecords pr;

    public Patient(String name, String s) {
        this.name = name;
        this.socialsecuritnumber = s;
        pr = null;
    }

    public void setPatientRecords(PatientRecords r) {
        this.pr = r;
    }
}
```

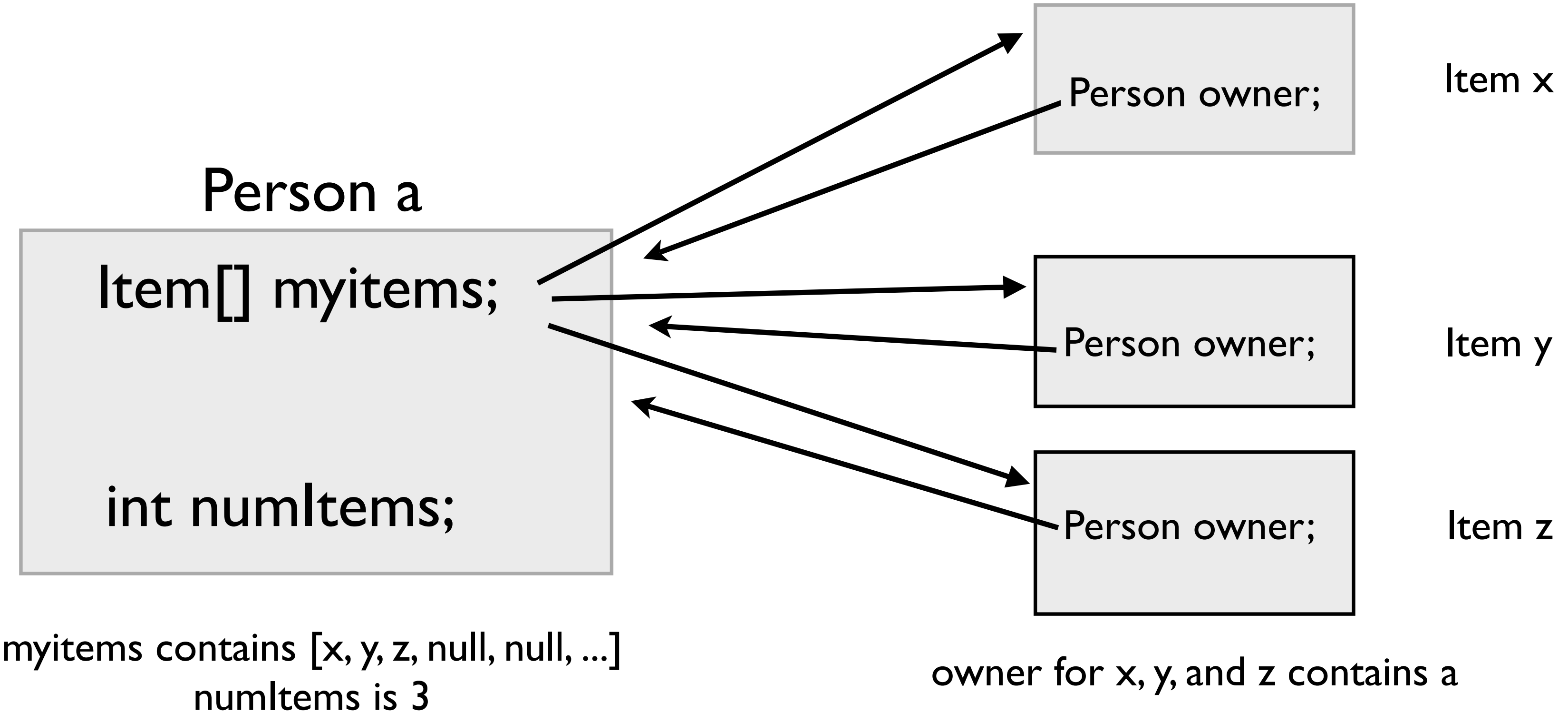
```
public class PatientRecords {
    private String doctor;
    private Patient p;


    public PatientRecords(String doctor) {
        this.doctor = doctor;
        p = null;
    }

    public void setPatient(Patient p) {
        this.p = p;
    }
}
```

# One-to-many relations

- Use an array or a list. For now, we will use an array, so we get practice with them, though lists are a better choice



# Initializing an array

- Initialize in the constructor
  - myitems = new Item[MAXITEMS];
- Creates an empty array of pointers to items
- Each pointer is initialized to empty, indicated by a value of null by Java.

# Using an array to store items

- Operations:
  - adding an item owned
  - removing an owned item: no longer owned
- The array will contain the items owned, but which slots contain the items?

- One design: Maintain the following invariants
  - Slots containing the items are at the beginning of the array
  - Unused slots at the end

Good: [x, y, z, *, *, *], numItems = 3
Bad: [*, x, *, y, z, *], numItems = 3

* is don't care. Good to set it to null.

# Adding an item

- Simply add at the end
  - items[numItems] = newitem;
- Why it works? Because of the invariant on the last slide

Initial: [x, y, z, *, *, *], numItems = 3

Adding w results in:

[x,y, z, w, *], numItems = 4

# Removing an item

- The item being removed can be anywhere in the array
- Need to find it first by scanning the array
- Then, to maintain the invariant, you need to shift the following elements to the left by 1
- Finally, decrement numItems

Initial: [x, y, z, *, *, *], numItems = 3

To delete y, first find its position, which is 1. Deleting with k = 1 results in

Final: [x,z, *, *, *], numItems = 2

# Shifting elements

- Example: deleting item at position k

```
for (int i = k; i < size-1; i++) {
    items[i] = items[i+1];
}
numItems--; // IMPORTANT. Re-establish invariant
items[numItems] = null;  // OPTIONAL. Good to do so.
```

Initial: [x, y, z, *, *, *], numItems = 3

Deleting with k = 1 results in

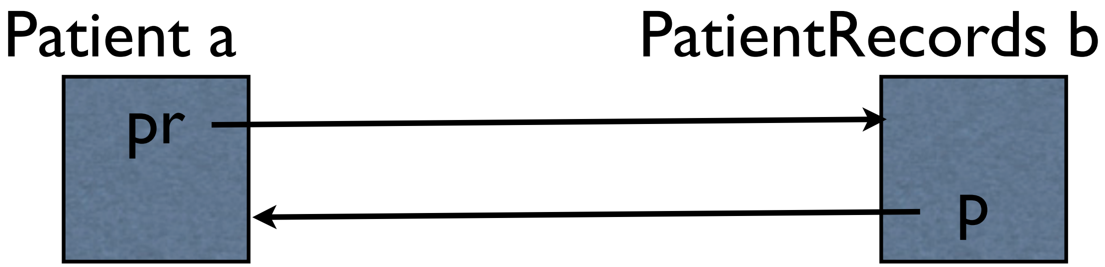
[x,z, *, *, *], numItems = 2

# Creating Relationships

- One way: create records, link them

```
public class Main {

    public static void main(String[] args) {
        Patient a = new Patient("Joe", "123-45-6789");
        PatientRecords b = new PatientRecords("dr. evans");
        a.setPatientRecords(b); // patient has a link to its record
        b.setPatient(a);         // record has a link to its patient
    }
}
```



# Problem

Desired invariant: two-way relationship

- Should not be possible for a user of these two classes to violate the above. Unfortunately, it is possible to do so.

```
 Patient a = new Patient(...);
 PatientRecords b = new PatientRecords("dr. evans");
 a.setPatientRecords(b); // patient has a link to its record
// no link created from b to a
```

# Better Solution

- The method that adds one relationship also adds the opposite relationship.

PatientRecords code

```
public void setPatient(Patient patient) {
    if (patient != p) {
        this.p = patient;
        patient.setPatientRecords(this);
    }
}
```

Patient code

```
public void setPatientRecords(PatientRecords r) {
    if (r != pr) {
        this.pr = r;
        r.setPatient(this);
    }
}
```

Invariant is maintained irrespective of whether setPatient or setPatientRecords is called

# Question

- Why are the if checks important? What happens if you omit them?

## PatientRecords code

```
public void setPatient(Patient patient) {
     if (patient != p) {
          this.p = patient;
          patient.setPatientRecords(this);
     }
}
```

## Patient code

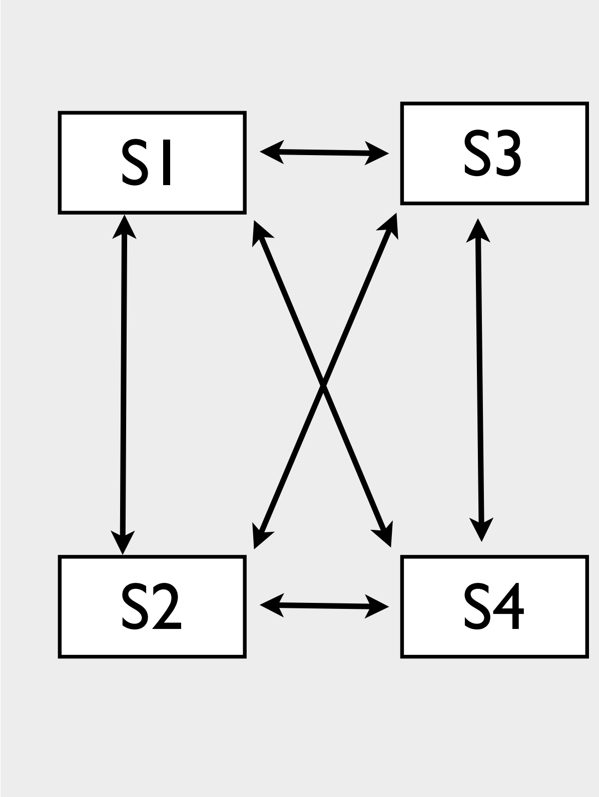
```
public void setPatientRecords(PatientRecords r) {
     if (r != pr) {
          this.pr = r;
          r.setPatient(this);
     }
}
```

# Another Example

- Class: Student
- Relationship: students can team up. A student can be in the same team as another student.
- One solution: Each student objects contains a list or array containing its team members
    - Student[] teamMembers;

# Reflect on the Design

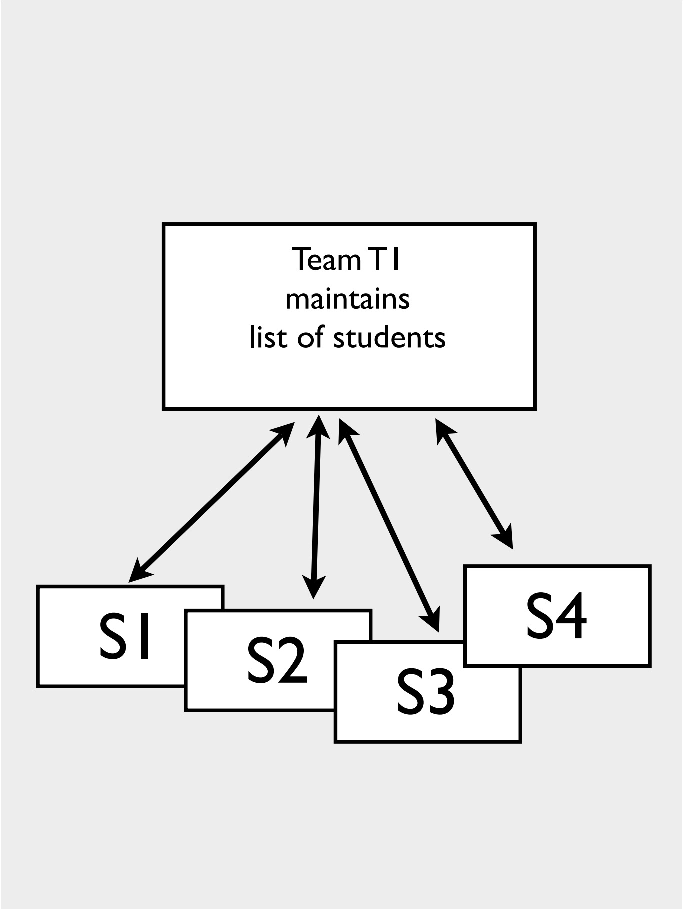
- Information in the current design is duplicated
- If 4 students are in a team, they have basically the same list. Updates must occur in 4 objects



Can we avoid duplication? Right now, we need to remember to update in multiple places to maintain the team invariant

# Solution

- Introduce another class to hold the relationship
  - A Student object can contain a link to its Team object
  - A Team object that contains links to all the team members in one array
- Now, the list of team members is in one place. Updates are easier



# What did we do?

- When information is duplicated in multiple objects, consolidate it one object
- Have all the objects share a single copy of that object by maintaining a link to it
- This is called introducing *indirection*. Rather than storing the information directly in multiple places, store it once and refer to it indirectly via a link

# Summary

- Objects can have relationships among themselves
- Use pointers or links for that
- Enforce invariants if links are bi-directional
- Avoid data duplication. If information is duplicated in multiple places, introduce
  - an additional class to hold the data in one place
  - Existing objects point to the object from new class