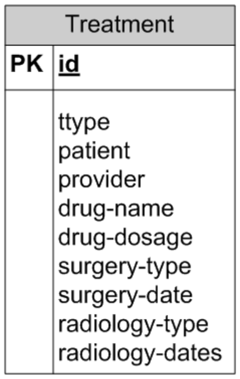
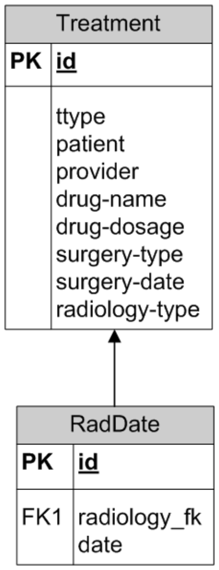
Single Table Strategy



149

149

Single Table Strategy



150

150

Single Table Strategy

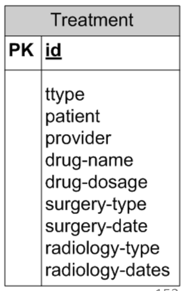
ID	TTYPE	PATIENT	PROVI DER	DRUG-NAME	DRUG-DOSAGE	SURGERY-DATE	RADIOL-DATES
1234	D	8907	5643	Prednisone	20mg		
5678	S	9076	3412			4-02-2011	

151

151

Single Table Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TTYPE")
public abstract class Treatment
      implements Serializable {
  private long id;
  private String treatmentType;
  @Column(name="TTYPE", length=2)
  public String getTreatmentType() {
      return treatmentType;
  }
  …
}
```



152

152

Single Table Strategy

```
@Entity
@DiscriminatorValue(“D”)
public class DrugTreatment extends Treatment {
   private String drug;
   private int dosage;
   …
}
@Entity
@DiscriminatorValue(“S”)
public class Surgery
        extends Treatment {
   private String type;
   private Date date;
   …
}
```

Treatment	
PK	id
	ttype patient provider drug-name drug-dosage surgery-type surgery-date radiology-type radiology-dates

153

153

Single Table Strategy

```
@Entity
@DiscriminatorValue(“R”)
public class Radiology extends Treatment {
   private String type;
   private Date date;
   …
}
```

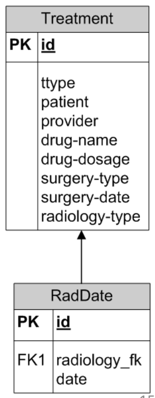
Treatment	
PK	id
	ttype patient provider drug-name drug-dosage surgery-type surgery-date radiology-type radiology-dates

154

154

Single Table Strategy

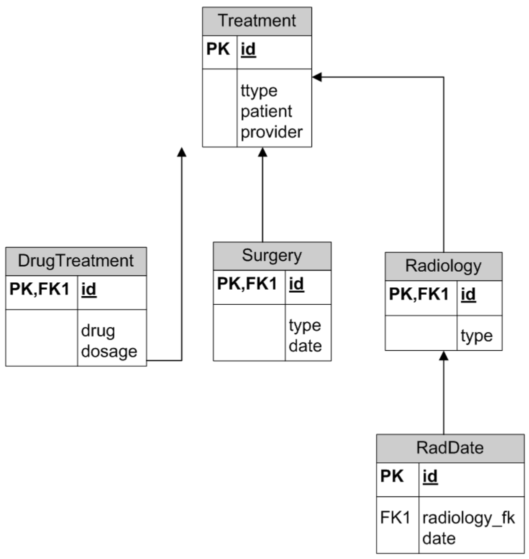
```
@Entity
@DiscriminatorValue(“R”)
public class Radiology extends Treatment {
  private String type;
  @OneToMany
  private Set<RadDate> dates;
  …
}
@Entity
public class RadDate {
  private Date date;
  …
}
```



155

155

Joined Table Strategy



156

156

Joined Table Strategy

ID	TTYPE	PATIENT	PROVIDER
1234	D	8907	5643
5678	S	9076	3412

ID	DRUG-NAME	DRUG-DOSAGE
1234	Prednisone	20mg

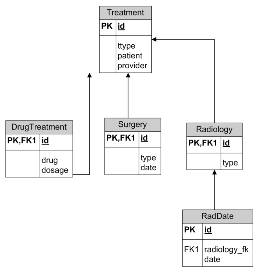
ID	SURGERY-DATE
5678	4-02-2011

157

157

Joined Table Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.JOIN_TABLE)
@DiscriminatorColumn(name="TTYPE")
public abstract class Treatment
      implements Serializable {
  private long id;
  private String treatmentType;
  @Column(name="TTYPE", length=2)
  public String getTreatmentType() {
      return treatmentType;
  }
  …
}
```



158

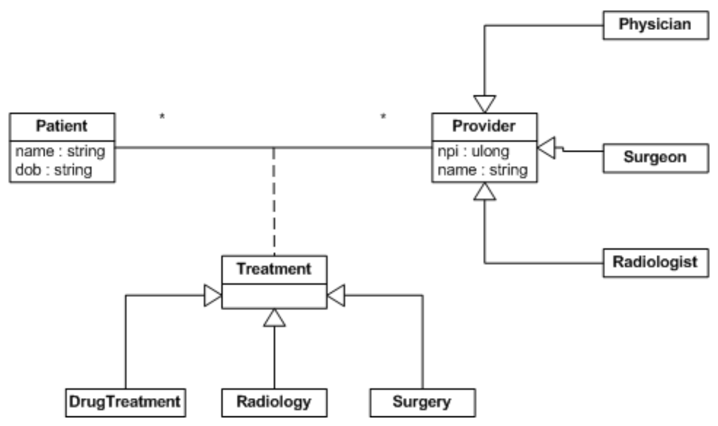
158

Summary

- Single Table Strategy:
  - Best for reads and updates
  - Wasted space in the table
- Joined Table Strategy:
  - Cost of joins for queries

AGGREGATE PATTERNS

Example: UML Diagram



161

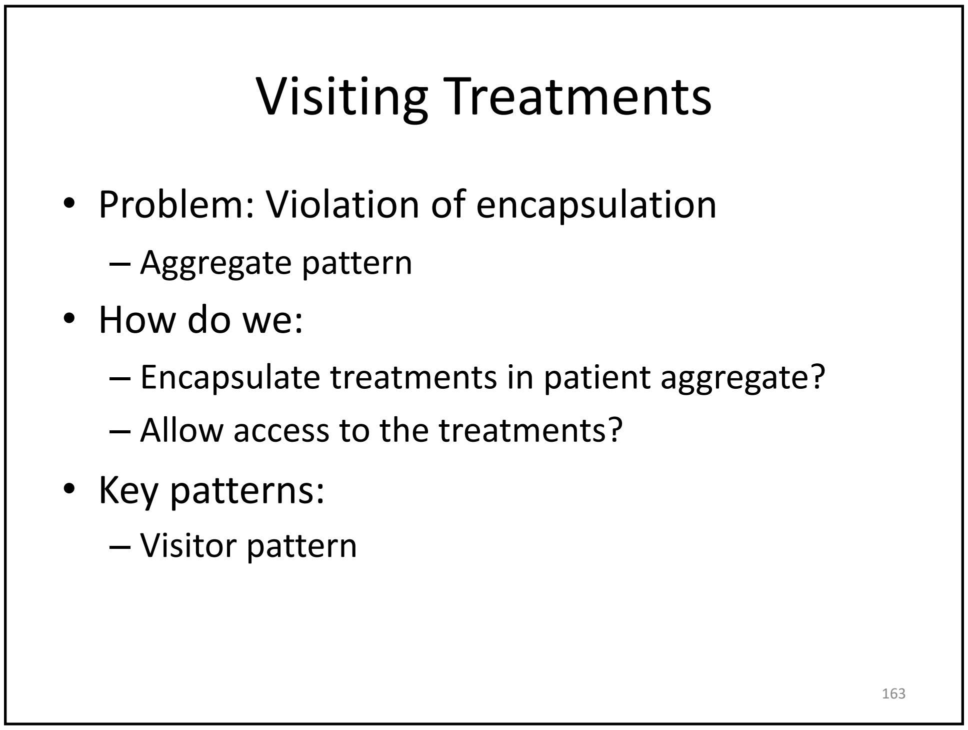
161

Visiting Treatments

```
Patient patient;
Set<Treatment> ts = patient.getTreatments();
for (Treatment t : ts)
  if (t instanceof DrugTreatment) {
     DrugTreatment dt = (DrugTreatment)t;
     … t.drug … t.dosage …
  } else if (t instanceof Surgery) {
     Surgery s = (Surgery)t;
     … s.date …
  } else if (t instanceof Radiology) {
     Radiology r = (Radiology)t;
     … r.dates …
  }
```

162

162

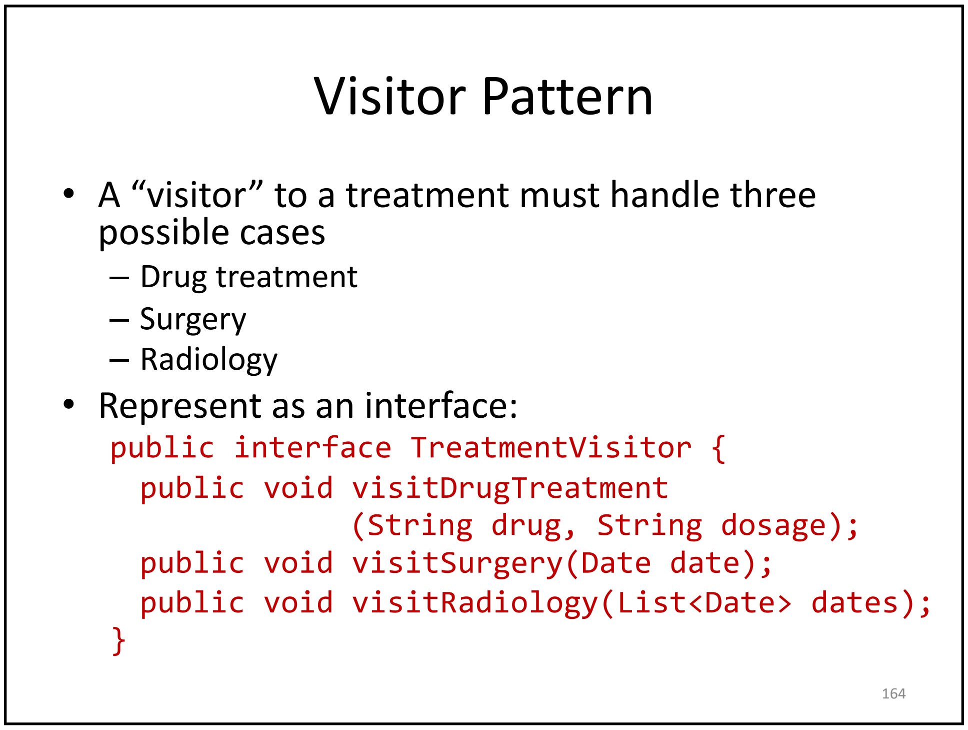


Visiting Treatments

- Problem: Violation of encapsulation
  - Aggregate pattern
- How do we:
  - Encapsulate treatments in patient aggregate?
  - Allow access to the treatments?
- Key patterns:
  - Visitor pattern

163

163



Visitor Pattern

- A “visitor” to a treatment must handle three possible cases
  - Drug treatment
  - Surgery
  - Radiology
- Represent as an interface:

```
public interface TreatmentVisitor {
  public void visitDrugTreatment
              (String drug, String dosage);
  public void visitSurgery(Date date);
  public void visitRadiology(List<Date> dates);
}
```

164

164

Visitor Pattern

- A base class defines a visitor method signature:

```
public abstract class Treatment {
  public abstract void visit(TreatmentVisitor v);
  …
}
```

- A concrete subclass dispatches the appropriate case:

```
public class DrugTreatment extends Treatment {
  @Override
  public void visit(TreatmentVisitor v) {
     v.visitDrugTreatment(this.drug, this.dosage);
  }
}
```

165

165

Visitor Pattern

- Patient aggregate dispatches visitor

```
@Entity
public class Patient {
  @OneToMany
  private Set<Treatment> treatments;
  public void visitTreatments
                    (TreatmentVisitor v) {
    for (Treatment t : treatments) t.visit(v);
  }
}
```

166

166

Aggregate Pattern

- Problem: Iteration over treatments should be client-controlled
  - Not aggregate-controlled
- Return a list of treatment identifiers
- Visit a treatment through patient API

167

167

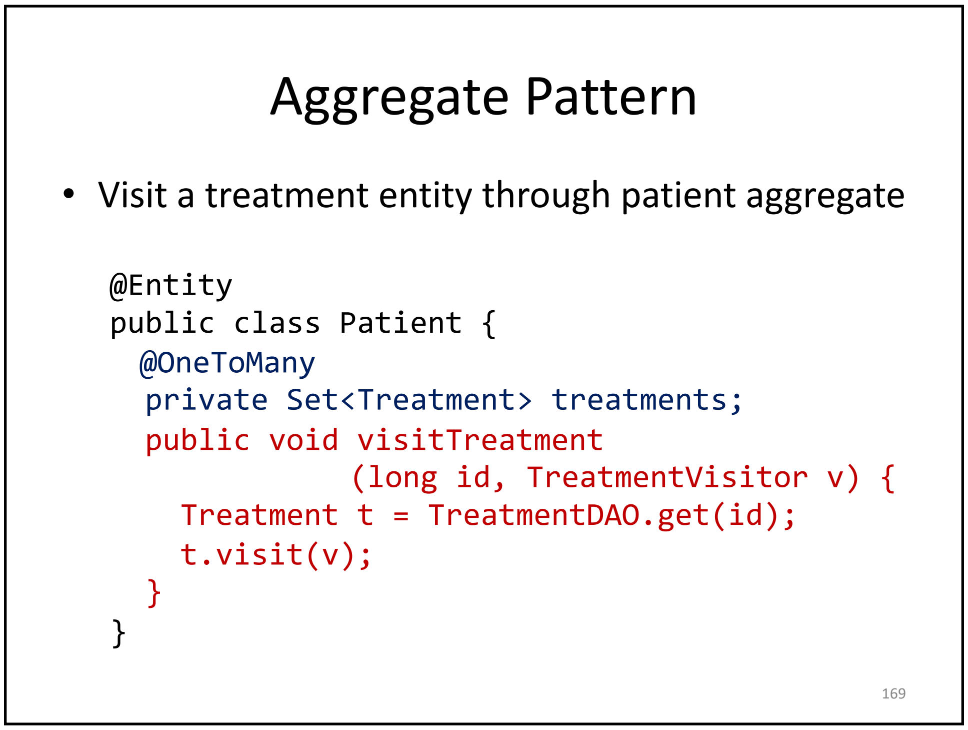
Aggregate Pattern

- Patient aggregate returns list of treatment ids

```
@Entity
public class Patient {
  @OneToMany
  private Set<Treatment> treatments;
  public List<Long> getTreatmentIds() {
    ArrayList<Long> ids = new ArrayList<Long>();
    for (Treatment t : treatments)
      ids.add(t.getId());
    return ids;
  }
}
```

168

168



Aggregate Pattern

- Visit a treatment entity through patient aggregate

```
@Entity
public class Patient {
  @OneToMany
  private Set<Treatment> treatments;
  public void visitTreatment
                (long id, TreatmentVisitor v) {
    Treatment t = TreatmentDAO.get(id);
    t.visit(v);
  }
}
```

169

169