OBJECT RELATIONAL MAPPING (ORM)

What's the problem?

- Need to persist entity state to a database
- One option: application control
  - Tricky for complex data structures
  - When to retrieve/save related entities
- Alternative: persistence handled by runtime
  - DB records saved/loaded as objects
  - Relationships represented as pointers
  - Retrieval/saving performed automatically
  - Java Persistence Architecture (JPA)

What's the problem?

- Need to persist entity state to a database
- One option: application control
  - Tricky for complex data structures
  - When to retrieve/save related entities
- Alternative: persistence handled by runtime
  - DB records saved/loaded as objects
  - Relationships represented as pointers
  - Retrieval/saving performed automatically
  - Java Persistence Architecture (JPA)

94

94

PERSISTENT DATA OBJECT (PDO)

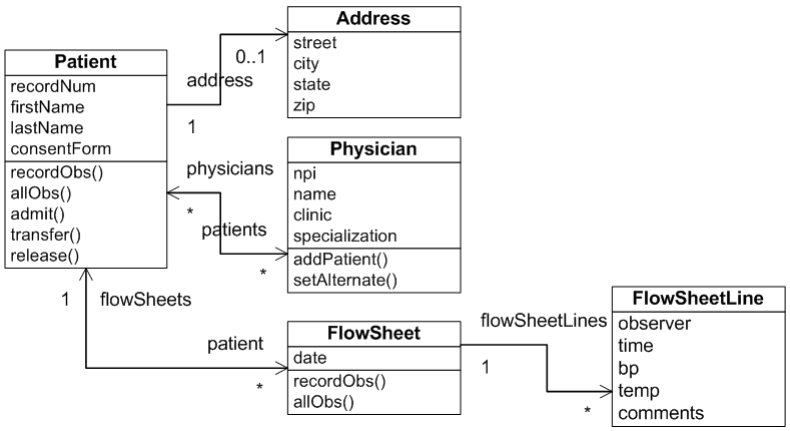
95

95

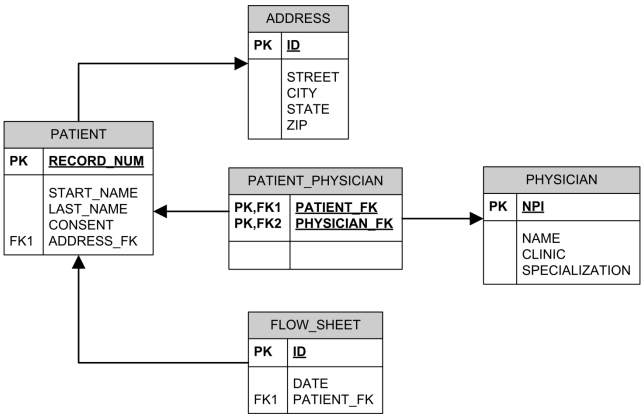
Patterns for Persistence

- Persistence Data Object (PDO):
  - Used to persist a domain entity object in the database
- Data Access Object (DAO):
  - Encapsulates and abstracts logic for data access and storage
  - Unnecessary with EntityManager?

Example



Example



98

98

Entities

- Lightweight persistent domain object
  - Object = Row in a DB table
  - Instance variables = Columns
- Entity class annotated with `@Entity`
  - Instance fields cannot be public
  - Getter/setter methods

```
T getProperty ()
void setProperty (T x)
```

99

99

Primary Keys

- Every entity must have a primary key
  - Simple primary key (`@Id`)
  - Composite primary key (`@EmbeddedId`, `@IdClass`)
- Key Generation: choices
  - `Assigned`: by application
  - `Sequence`: generate unique values
  - `Identity`: auto increment
  - `Table`: separate PK table

100

100

Patient Class

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
   private long recordNum;
   private String firstName;
   private String lastName;

   @id(name = "RECORD_NUM")
   @GeneratedValue
   public int getRecordNum() {
       return this.recordNum;
   }
   public void setRecordNum(int r) {
       this.recordNum = r;
   }
```

101

101

Patient Class

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
   private long recordNum;
   private String firstName;
   private String lastName;

   @id(name = "RECORD_NUM")
   @GeneratedValue
   public int getRecordNum() {
       return this.recordNum;
   }
   public void setRecordNum(int r) {
       this.recordNum = r;
   }
```

102

102

Patient Class

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
   …
   @Column(name="FIRST_NAME")
   public String getFirstName() { return this.firstName; }
   public void setFirstName(String firstName) {
       this.firstName = firstName;
   }
   @Column(name="LAST_NAME")
   public String getLastName() { return this.lastName; }
   public void setLastName(String lastName) {
       this.lastName = lastName;
   }
```

103

103

Patient Class

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
   …
   @Lob
   private byte[] consentForm;

   @OneToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "ADDRESS_FK", nullable = true)
   private Address address;
   @OneToMany(cascade = CascadeType.ALL, mappedBy =
   "patient")
   private List<FlowSheet> flowsheets;
   @ManyToMany(mappedBy = "patient")
   private Set<Physician> physicians;
   ...
}
```

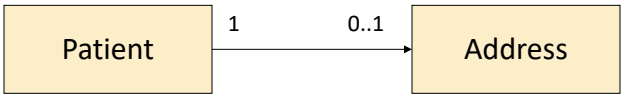

104

ENTITY RELATIONSHIPS (1/2)

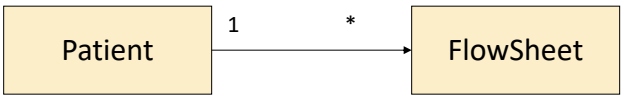
105

Entity Relationships

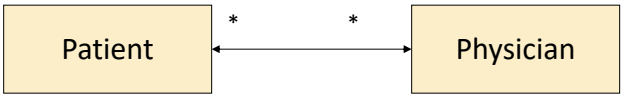
One-to-one (1:1)



One-to-many (1:*)



Many-to-many (*:*)



106

106

Entity Relationship Directions

- Relationships can be:
  - Unidirectional (owning side)
  - Bidirectional (owning & inverse sides)
- Owning side
  - For 1:n or n:1, entity where foreign key is stored
  - For 1:1, contains the foreign key
  - For n:n relations, either side
  - Owning side determines the updates to the relationships in the database

107

107

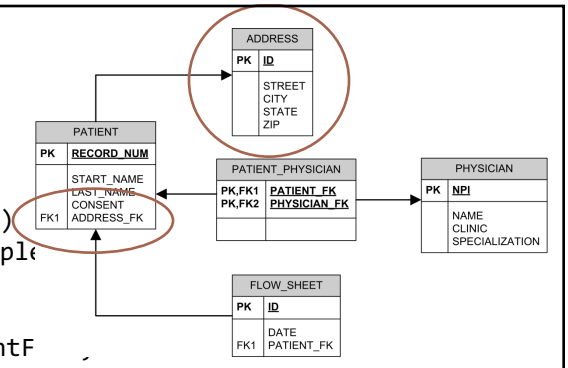
Annotations

- Annotations
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- mappedBy Attribute
  - Inverse side must refer to owning side
  - mappedBy cannot be specified on the ManyToOne annotation

108

108

Patient Class



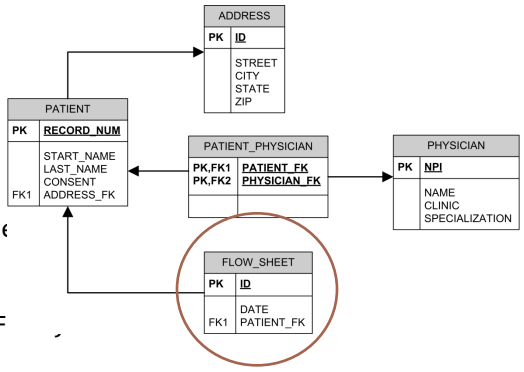

```
@Entity
@Table(name = "PATIENT")
public class Patient imple
   …
   @Lob
   private byte[] consentF

   @OneToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "ADDRESS_FK", nullable = true)
   private Address address;
   @OneToMany(cascade = CascadeType.ALL,
              mappedBy = "patient")
   private List<FlowSheet> flowsheets;
   @ManyToMany(mappedBy = "patient")
   private Set<Physician> physicians;
   ...
}
```

109

109

Patient Class



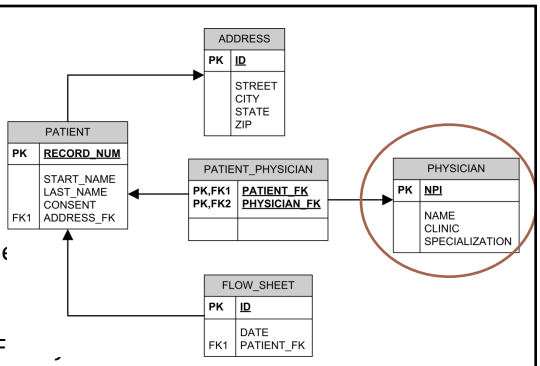

```
@Entity
@Table(name = "PATIENT")
public class Patient imple
   …
   @Lob
   private byte[] consentF

   @OneToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "ADDRESS_FK", nullable = true)
   private Address address;
   @OneToMany(cascade = CascadeType.ALL,
              mappedBy = "patient")
   private List<FlowSheet> flowsheets;
   @ManyToMany(mappedBy = "patient")
   private Set<Physician> physicians;
   ...
}
```

110

110

Patient Class




```
@Entity
@Table(name = "PATIENT")
public class Patient imple
   …
   @Lob
   private byte[] consentF

   @OneToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "ADDRESS_FK", nullable = true)
   private Address address;
   @OneToMany(cascade = CascadeType.ALL,
              mappedBy = "patient")
   private List<FlowSheet> flowsheets;
   @ManyToOne(mappedBy = "patients")
   private Set<Physician> physicians;
   ...
}
```

111

111

Address class

```
@Entity
@Table(name = "ADDRESS")
public class Address implements Serializable {
  @Id @GeneratedValue
  private long id;
  private Street street;
  private City city;
  private State state;
  private Zip zip;
}
```

112

112

Flow Sheet Class

```
@Entity
@Table(name = "FLOW_SHEET")
public class FlowSheet implements Serializable {
  @Id @GeneratedValue
  private long id;
  @Temporal(TemporalType.DATE)
  private Date date;
  List<FlowSheetLine> lines;
  ...
  @ManyToOne
  @JoinColumn(name = "PATIENT_FK", nullable = false)
  Patient patient;
}
```

113

113

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
       …
       @OneToMany(cascade = CascadeType.ALL,
                  mappedBy = "patient")
       private List<FlowSheet> flowsheets;
       ...
}
```

```
@Entity
@Table(n
public c
  @Id @G
  privat
  @Tempo
  private Date date;
  List<FlowSheetLine> lines;
  ...
  @ManyToOne
  @JoinColumn(name = "PATIENT_FK", nullable = false)
  Patient patient;
}
```

114

114

ENTITY RELATIONSHIPS (2/2)

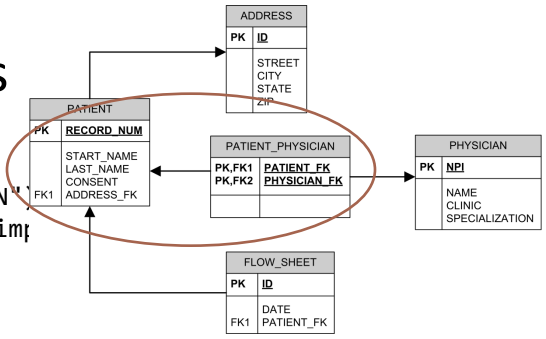
115

115

Physician Class

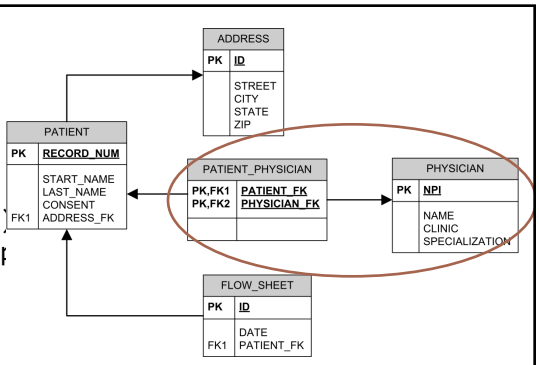
```
@Entity
@Table(name = "PHYSICIAN")
public class Physician implements Serializable {
   @Id
   @Column(name = "NPI")
   private long npi;
   ...
   @ManyToMany
   @JoinTable(name = "PATIENT_PHYSICIAN",
        joinColumns = @JoinColumn (name = "PATIENT_FK",
                      referencedColumn = "RECORD_NUM"),
        inverseJoinColumns =
                      @JoinColumn (name = "PHYSICIAN_FK",
                      referencedColumn = "NPI"))
   private Collection<Patient> patients;
}
```


Physician Class




```
@Entity
@Table(name = "PHYSICIAN")
public class Physician implements Serializable {
   @Id
   @Column(name = "NPI")
   private long npi;
   ...
   @ManyToMany
   @JoinTable(name = "PATIENT_PHYSICIAN",
        joinColumns = @JoinColumn (name = "PATIENT_FK",
                      referencedColumn = "RECORD_NUM"),
        inverseJoinColumns =
                      @JoinColumn (name = "PHYSICIAN_FK",
                      referencedColumn = "NPI"))
   private Collection<Patient> patients;
}
```

Physician Class




```
@Entity
@Table(name = "PHYSICIAN")
public class Physician implements Serializable {
   @Id
   @Column(name = "NPI")
   private long npi;
   ...
   @ManyToMany
   @JoinTable(name = "PATIENT_PHYSICIAN",
        joinColumns = @JoinColumn (name = "PATIENT_FK",
                      referencedColumn = "RECORD_NUM"),
        inverseJoinColumns =
                      @JoinColumn (name = "PHYSICIAN_FK",
                      referencedColumn = "NPI"))
   private Collection<Patient> patients;
}
```


118

```
@Entity
@Table(name = "PATIENT")
public class Patient implements Serializable {
       …
       @ManyToMany(mappedBy = "patients")
       private Set<Physician> physicians;
       ...
}
```

```
@Entity
@Table(name =
public class
   @Id
   @Column(name = "NPI")
   private long npi;
   ...
   @ManyToMany
   @JoinTable(name = "PATIENT_PHYSICIAN",
        joinColumns = @JoinColumn (name = "PATIENT_FK",
                      referencedColumn = "RECORD_NUM"),
        inverseJoinColumns =
                      @JoinColumn (name = "PHYSICIAN_FK",
                      referencedColumn = "NPI"))
   private Collection<Patient> patients;
}
```


119

Database Tables

One-to-many (1:*)


```
create table PATIENT {
   RECORD_NUM  integer,
   FIRST_NAME      varchar(32),
   LAST_NAME       varchar(32)
);
```

```
create table FLOW_SHEET {
   ID                  integer,
   PATIENT_FK   integer),
);
```

120

120

Database Tables

One-to-many (1:*)


```
create table PATIENT {
   RECORD_NUM  integer,
   FIRST_NAME      varchar(32),
   LAST_NAME       varchar(32)
);
```

```
create table FLOW_SHEET {
   ID                  integer,
   PATIENT_FK   integer),
);
```

121

121

Example: Patient Flow Sheet

- Owner:
  - Flow sheet is the owner, thus after changes sheet has to be merged
  - If patient attributes have changed as well, then those are not automatically stored
    - Merge patient individually
    - Declare cascade=MERGE option on getPatient method

Removing Entities

- Cascade
  - When a patient is removed, their flow sheets should also be removed
    - `Cascade=REMOVE`
    - `Cascade=ALL`
  - Remove

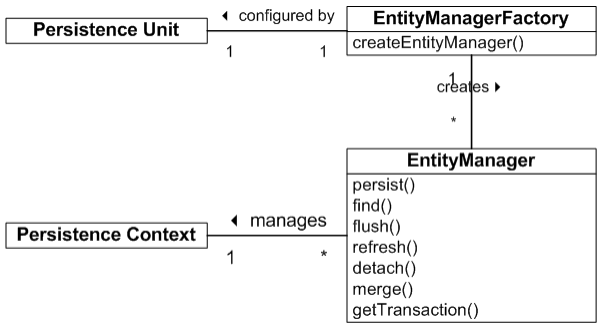
```
Patient p = em.find(Patient.class, id);
em.remove(p);
```

ENTITY MANAGER

124

124

JPA API



125

125

Entity Manager

- Purpose
  - Manages the entity instance lifecylce
  - Methods to interact with the *persistence context*
- Persistence Context
  - A set of entity instances in which for any entity identity there is a unique entity instance
- Dependency Injection
  - Container-managed entity manager is injected by the container

126

126

Entity Manager

- Application-managed EntityManager:

```
EntityManagerFactory emf =
   Persistence.createEntityManagerFactory();
EntityManager em =
   emf.createEntityManager("PatientPU");
PatientDAO repository = new PatientDAO(em);
```

- Container-managed EntityManager:

```
@PersistenceContext(unitName = "PatientPU")
EntityManager em;
```

127

127

Entity Manager

- Application-managed EntityManager:

```
EntityManagerFactory emf =
   Persistence.createEntityManagerFactory();
EntityManager em =
   emf.createEntityManager("PatientPU");
PatientDAO repository = new PatientDAO(em);
```

- Container-managed EntityManager:

```
@PersistenceContext(unitName = "PatientPU")
EntityManager em;
```

128

128

DATA ACCESS OBJECT (DAO)

129

129

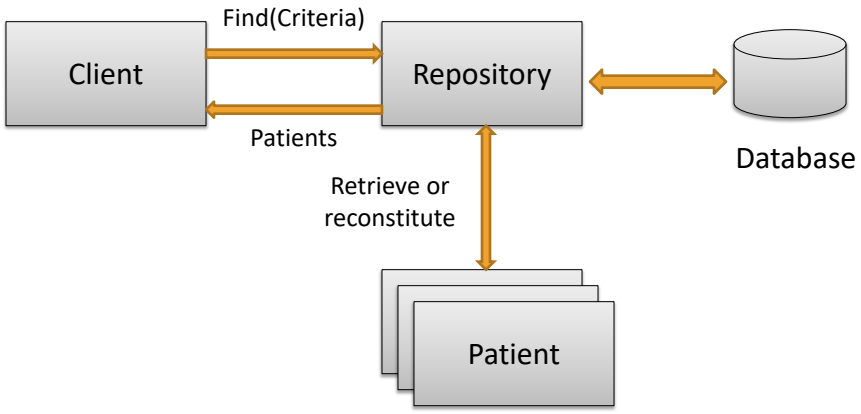
Patterns for Persistence

- Persistence Data Object (PDO):
  - Used to persist a domain entity object in the database
- Data Access Object (DAO):
  - Encapsulates and abstracts logic for data access and storage
  - Unnecessary with EntityManager? Antithesis of PDO

130

130

Repository (DAO) Pattern



131

131

DAO Example

```
public class PatientDAO {
  protected EntityManager em;
  public PatientDAO(EntityManager em) {
      this.em = em;
  }
  public Patient create
      (String first, String last, byte[] consent) {
      Patient patient = new Patient ();
      patient.setFirstName(first);
      patient.setLastName(last);
      patient.setConsentForm(consent);
      em.persist(patient);
      return patient;
  }
  …
```

132

132

DAO Example

```
@NamedQuery(name = "findByName",
  query = "select p from Patient p where
      p.firstName = :first and p.lastName = :last")
public class PatientDAO {
  …
  public List<Patient> findByName
      (String firstName, String lastName, int max) {
      Query query =
            em.createNamedQuery("findByName")
                  .setParameter("first", firstName)
                  .setParameter("last", lastName)
                  .setMaxResults(max)
      return (List<Patient>) query.getResultList();
  }
```

133

133

DAO Example

```
public class PatientDAO {
  …
  public void delete(long id) {
      Patient patient = em.find(Patient.class, id);
      if (patient != null) em.remove(patient);
  }
  void beginTransaction() {
      em.getTransaction().begin();
  }
  void endTransaction() {
      em.getTransaction().commit();
  }
}
```

134

134

Other EM Operations

- `flush()`: flush updates to the DB
- `refresh()`: roll back changes not committed to DB
- `detach()`: object is no longer “managed”
- `merge()`: re-attach object to “managed” objects

135

135