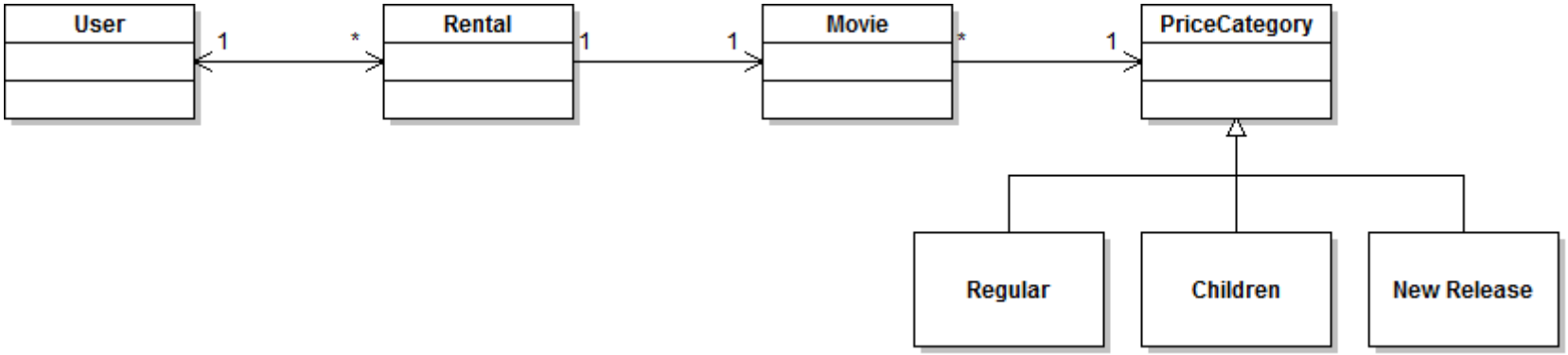
# Movie Rental (Assignment 2-4)

- **Entity Definitions**
- **Repository Definitions**
- **Named Queries**

# Movie Rental Application

- **Model Classes**



# Movie Rental Entities: User

```
@Entity
@Table(name = "USERS")
public class User {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "USER_ID")
   private Long id;

   @Column(name = "USER_NAME")
   private String lastName;

   …

   @OneToMany(mappedBy = "user",
      cascade = CascadeType.REMOVE)
   private List<Rental> rentals;

   protected User() { }
```

Inverse side of a bidirectional association

rentals are deleted if user is deleted (composition)

# Movie Rental Entities: Rental

```
@Entity
@Table(name = "RENTALS")
public class Rental {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "RENTAL_ID")
   private Long id;

   @OneToOne
   @JoinColumn(name = "MOVIE_ID")
   private Movie movie;

   @ManyToOne
   @JoinColumn(name = "USER_ID")
   private User user;

   @Column(name = "RENTAL_RENTALDATE")
   private LocalDate rentalDate;
```

Unidirectional association between rental and movie

Owning side of user-rental bidirectional association

# Movie Rental Entities: Movie

```
@Entity
@Table(name = "MOVIES")
public class Movie {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "MOVIE_ID")
   private Long id;

   @Column(name = "MOVIE_TITLE")
   private String title;

   …

   @ManyToOne
   @JoinColumn(name = "PRICECATEGORY_FK")
   private PriceCategory priceCategory;

   protected Movie() { }
```

Unidirectional association between movie and price category

# Movie Rental Entities: PriceCategory

```
@Entity
@Table(name = "PRICECATEGORIES")
@DiscriminatorColumn(name = "PRICECATEGORY_TYPE")
public abstract class PriceCategory {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "PRICECATEGORY_ID")
   private Long id;
```

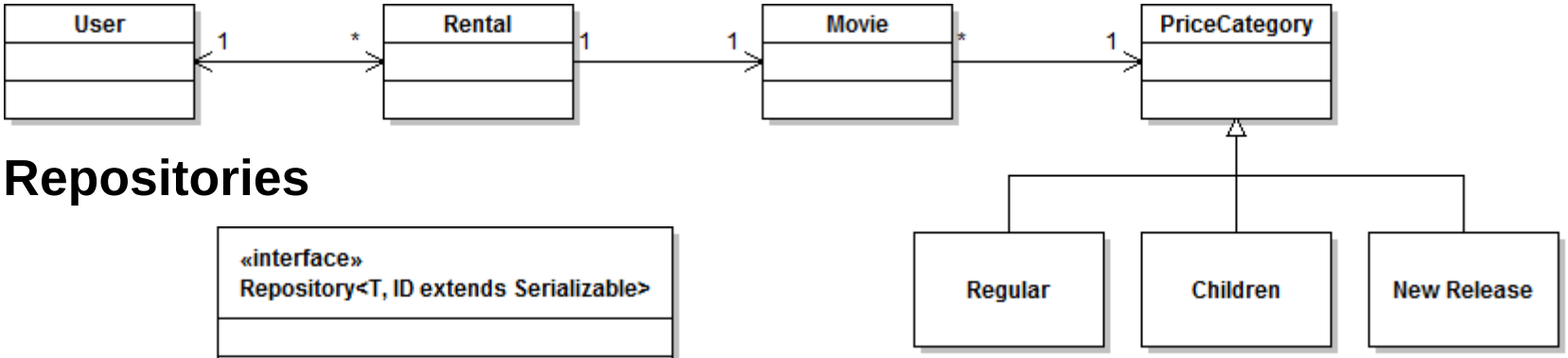
```
@Entity
@DiscriminatorValue("Children")
public class PriceCategoryChildren extends PriceCategory {
   @Override
   public double getCharge(int daysRented) { … }
   @Override
   public String toString() { … }
}
```

# Movie Rental (Assignment 2-4)

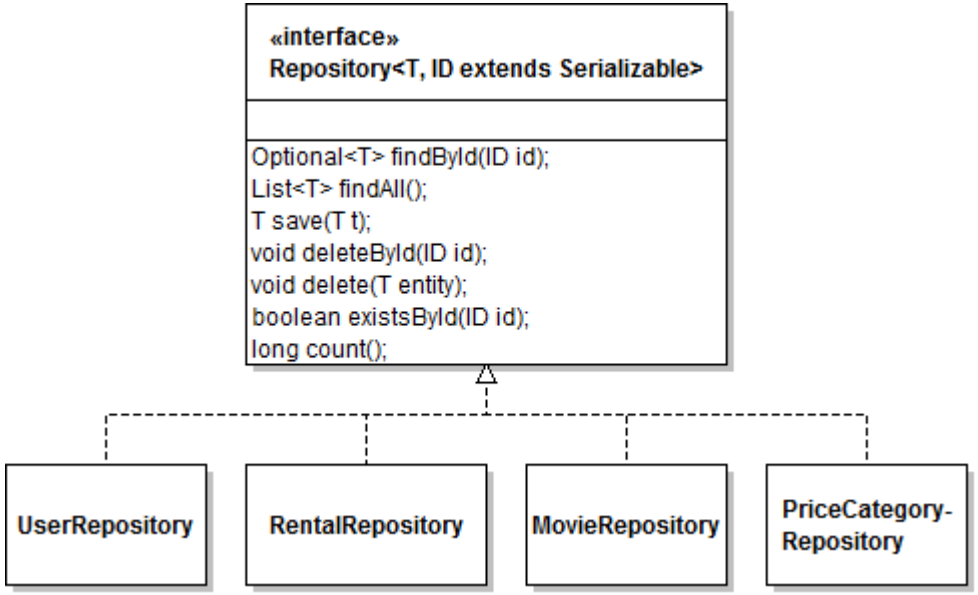
- **Entity Definitions**
- **Repository Definitions**
- **Named Queries**

# Movie Rental Application

- **Model Classes**




- **Repositories**



# Movie Rental Repositories: Abstract Repository

```
public abstract class AbstractJpaRepository<T>
                                        implements Repository<T, Long> {
   @PersistenceContext
   private EntityManager em;

   private final Class<T> type;

   protected AbstractJpaRepository(Class<T> type) {
      this.type = type;
   }



   @Override
   public Optional<T> findById(Long id) {
      return Optional.ofNullable(em.find(type, id));
   }
   …
```

**In JpaMovieRepository:**

```
public JpaMovieRepository() {
   super(Movie.class);
}
```

# Movie Rental Repositories: Abstract Repository

```
public abstract class AbstractJpaRepository<T>
                                        implements Repository<T, Long> {
   @PersistenceContext
   private EntityManager em;

   private final Class<T> type;

   protected AbstractJpaRepository() {
      type = (Class<T>) ((ParameterizedType)getClass()
                                        .getGenericSuperclass())
                  .getActualTypeArguments()[0];
   }

   @Override
   public Optional<T> findById(Long id) {
      return Optional.ofNullable(em.find(type, id));
   }
   …
```

# Movie Rental Repositories: Abstract Repository

```
    @Override
    public T save(T m) {
        return em.merge(m);
    }

    @Override
    public void deleteById(Long id) {
        em.remove(em.getReference(type, id));
    }

    @Override
    public void delete(T entity) {
        em.remove(em.merge(entity));
    }

}
```

Probably a distinction should be made between persist and merge (due to the cascade annotations)

`getReference` returns an instance whose state may be lazily fetched (a proxy)

Only managed entities can be removed, otherwise a "Removing a detached entity" exception is thrown

# Movie Rental Repositories: Movie Repository

```
@Repository
public class JpaMovieRepository
      extends AbstractJpaRepository<Movie> implements MovieRepository {

   @PersistenceContext
   private EntityManager em;

   @Override
   public List<Movie> findAll() {
      return em.createQuery("SELECT m FROM Movie m",
                                          Movie.class).getResultList();
   }

   @Override
   public long count() {
      return em.createQuery("SELECT COUNT(m) FROM Movie m",
                                          Long.class).getSingleResult();
   }
```

# Movie Rental Repositories: Movie Repository

```
   @Override
   public boolean existsById(Long id) {
      return em.createQuery(
            "SELECT COUNT(m) FROM Movie m where m.id = :id", Long.class)
         .setParameter("id", id)
         .getSingleResult() > 0;
   }

   @Override
   public List<Movie> findByTitle(String title) {
      return em.createQuery(
            "SELECT m FROM Movie m WHERE m.title = :title", Movie.class)
         .setParameter("title", title)
         .getResultList();
   }

}
```

# Movie Rental Repositories: Abstract Repository

```
public abstract class AbstractJpaRepository<T>
                                        implements Repository<T, Long> {
   @PersistenceContext
   private EntityManager em;
   private final Class<T> type;
   private final String name;

   protected AbstractJpaRepository() {
      type = (Class<T>) ((ParameterizedType)getClass()
          .getGenericSuperclass()).getActualTypeArguments()[0];
      name = type.getSimpleName();
   }

   @Override
   public List<T> findAll() {
      return em.createQuery(String.format("SELECT e FROM %s e", name),
           type).getResultList();
   }
```

If the entity name is the unqualified class name

# Movie Rental Repositories: Abstract Repository

```
   @Override
   public long count() {
      return em.createQuery(
           String.format("SELECT COUNT(e) FROM %s e", name), Long.class)
            .getSingleResult();
   }

   @Override
   public boolean existsById(Long id) {
      return em.createQuery(
         String.format("SELECT COUNT(e) FROM %s e where e.id = :id",
                                                   name), Long.class)
            .setParameter("id", id)
            .getSingleResult() > 0;
   }

}
```

# Movie Rental Repositories: Abstract Repository

```
public abstract class AbstractJpaRepository<T>
                                        implements Repository<T, Long> {
   @PersistenceContext
   private EntityManager em;
   private final Class<T> type;
   private final String name;

   protected AbstractJpaRepository() {
      type = (Class<T>) ((ParameterizedType)getClass()
          .getGenericSuperclass()).getActualTypeArguments()[0];

      Entity a = type.getAnnotation(Entity.class);
      if("".equals(a.name())) {
         name = type.getSimpleName();
      } else {
         name = a.name();
      }
   }
```

Uses the name from the @Entity definition (if defined)

# Movie Rental (Assignment 2-4)

- **Entity Definitions**
- **Repository Definitions**
- **Named Queries**

# Movie Rental: Named Queries

```
@Entity
@Table(name = "MOVIES")
@NamedQuery(name = Movie.FIND_ALL,
            query = "SELECT m FROM Movie m")
@NamedQuery(name = Movie.EXISTS,
            query = "SELECT COUNT(m) FROM Movie m where m.id = :id")
@NamedQuery(name = Movie.COUNT,
            query = "SELECT COUNT(m) FROM Movie m")
@NamedQuery(name = Movie.FIND_BY_TITLE,
            query = "SELECT m FROM Movie m WHERE m.title = :title")
public class Movie {
   public static final String FIND_ALL = "Movie.all";
   public static final String FIND_BY_TITLE = "Movie.byTitle";
   public static final String EXISTS = "Movie.exists";
   public static final String COUNT = "Movie.count";
   …
}
```

# Named Queries: Movie Repository

```
@Override
public List<Movie> findAll() {
   return em.createNamedQuery(Movie.FIND_ALL, Movie.class)
      .getResultList();
}

@Override
public boolean existsById(Long id) {
   return em.createNamedQuery(Movie.EXISTS, Long.class)
      .setParameter("id", id)
      .getSingleResult() > 0;
}

@Override
public List<Movie> findByTitle(String title) {
   return em.createNamedQuery(Movie.FIND_BY_TITLE, Movie.class)
      .setParameter("title", title)
      .getResultList();
}
```

# Named Queries: Abstract Repository

```
…
   private String getField(String name) {
      try { return (String) type.getField(name).get(null); }
      catch (Exception e) { throw new RuntimeException(e); }
   }

   @Override
   public List<T> findAll() {
      return em.createNamedQuery(getField("FIND_ALL"), type)
         .getResultList();
   }

   @Override
   public boolean existsById(Long id) {
      return em.createNamedQuery(getField("EXISTS"), Long.class)
         .setParameter("id", id)
         .getSingleResult() > 0;
   }
```