



Java Persistence API (JPA): ОСНОВЫ

Андрей Родионов
(на основе материалов Михаила Вайсмана)



Содержание

- Что такое и Почему используем O/R Mapper (ORM)?
- Что нам дает JPA?
- O/R Отображения
- Что такое Entity?
- Программная модель JPA
- EntityManager и операции управления Entity
- Жизненный цикл Entity

Почему Object/Relational Mapping?

- Одна из главных частей любого энтерпрайз приложения – уровень persistence
 - Доступ и управление перманентными данными, обычно с применением реляционной БД
- ORM берет на себя “превращение” таблицы в объект
 - Данные живут в реляционной БД, т.е. в таблицах (в строках и столбцах)
 - Мы же хотим работать с объектами, а не с колонками и столбцами

Что нам дает JPA?

- Упрощение модели persistens
 - > Использование значений по умолчанию вместо сложных настроек
 - > Отказ от конфигурационных файлов
- Предоставление легковесной модели persistence
 - > Увеличение быстродействия
- Единый API для Java SE и Java EE

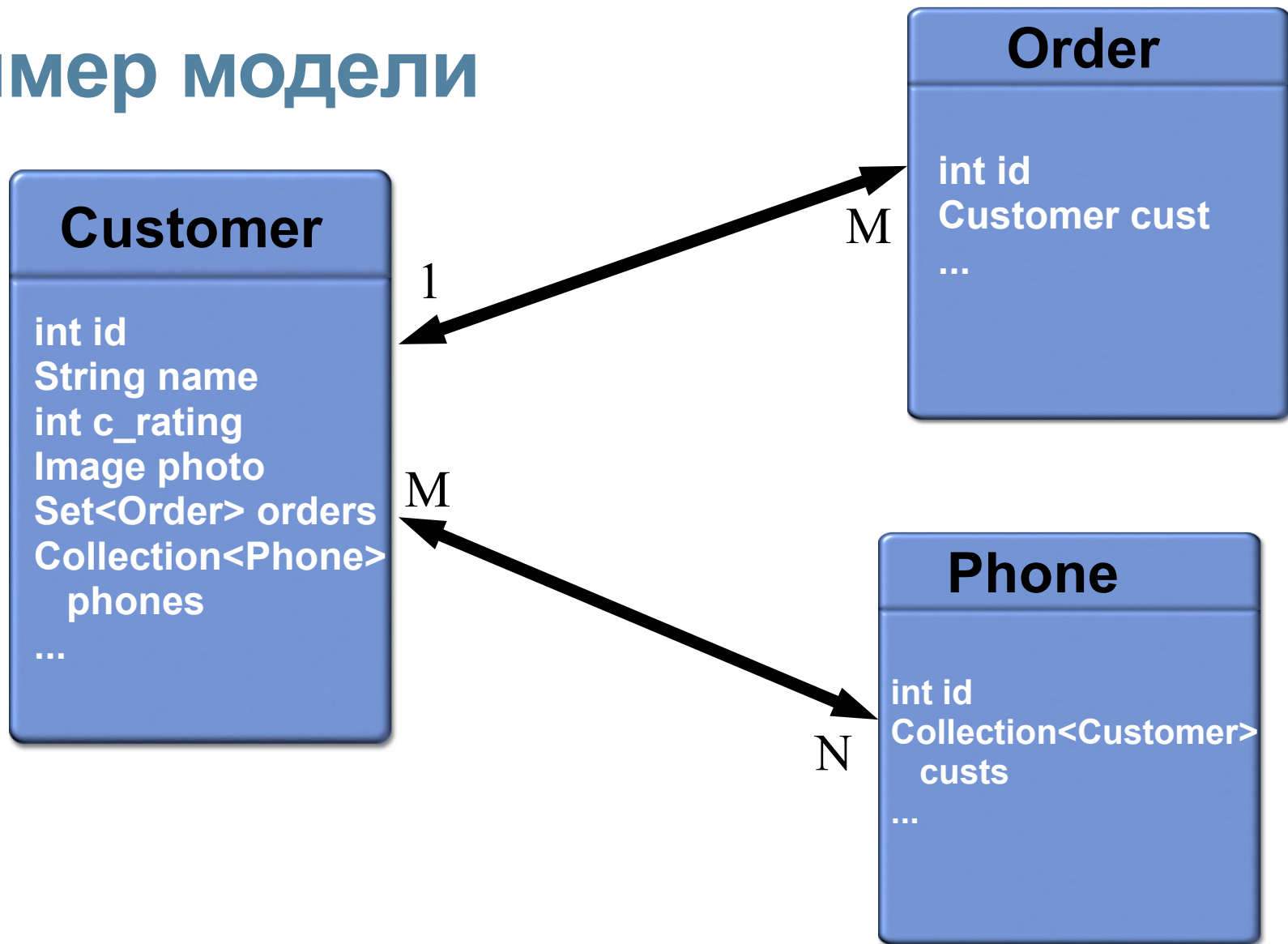
You'll need three artifacts to implement a JPA-compliant program

- An entity class
- A persistence.xml file
- A class through which you will insert, update, or find an entity

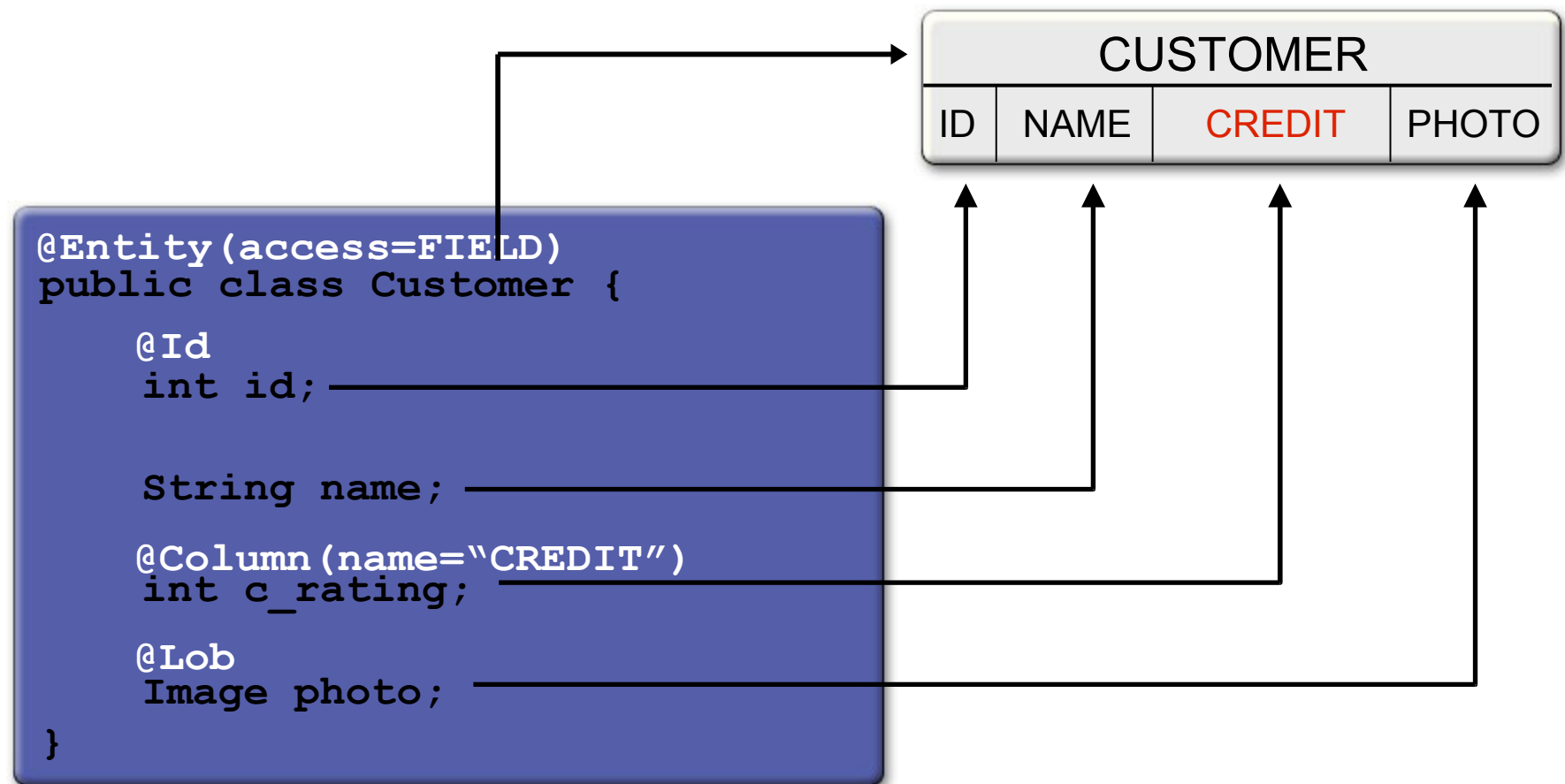
O/R Отображения

- Обширный набор аннотаций для описания отображений (mapping)
 - > Связи
 - > Объединения
 - > Таблицы и колонки БД
 - > Генераторы последовательностей для БД
 - > Многое другое
- Возможно использовать отдельный конфигурационный файл для описания отображений (mapping)

Пример модели



Пример отображения



Умолчания в действии:
специально описываются только
расхождения в именах.

Пример Entity

@Entity

```
public class Customer implements Serializable {  
    @Id protected Long id;  
    protected String name;  
    @Embedded protected Address address;  
    protected PreferredStatus status;  
    @Transient protected int orderCount;  
  
    public Customer() {}  
  
    public Long getId() {return id;}  
    protected void setId(Long id) {this.id = id;}  
  
    public String getName() {return name;}  
    public void setName(String name) {this.name = name;}  
  
    ...  
}
```

Идентификация Entity

- Любой Entity имеет перманентный идентификатор
 - > Он отображается в первичный ключ в таблице
- Идентификатор — примитивный тип
 - > `@Id`—одиночное поле/свойство в Entity классе
 - > `@GeneratedValue`—значение может генерироваться автоматически, используя различные стратегии (SEQUENCE, TABLE, IDENTITY, AUTO)
- Идентификатор – пользовательский класс
 - > `@EmbeddedId`—одиночное поле/свойство в Entity классе
 - > `@IdClass`—соответствует множеству полей в Entity классе

Программная модель JPA

- Entity это простой Java класс (POJO)
- Для описания класс как Entity используется аннотации

@Entity

```
public class Employee {  
    // Persistent/transient fields  
    // Property accessor methods  
    // Persistence logic methods  
}
```

Отображение отношений

- Аннотации для описания отношений между Entity
 - > @OneToOne
 - > @OneToMany
 - > @ManyToOne
 - > @ManyToMany

EntityManager

- Управляет жизненным циклом Entity объектов
 - > `persist()` - помещает объект в БД
 - > `remove()` - удаляет объект из БД
 - > `merge()` - синхронизирует с БД состояние отсоединенного объекта
 - > `refresh()` - обновляет из БД состояние объекта

Операция Persist

```
public Order createNewOrder(Customer customer) {  
    // Создаем новый объект  
    Order order = new Order(customer);  
  
    // После вызова метода persist() объект меняет свой  
    // статус на управляемый. Во время очередной  
    // операции записи в БД объект будет помещен в БД.  
    entityManager.persist(order);  
  
    return order;  
}
```

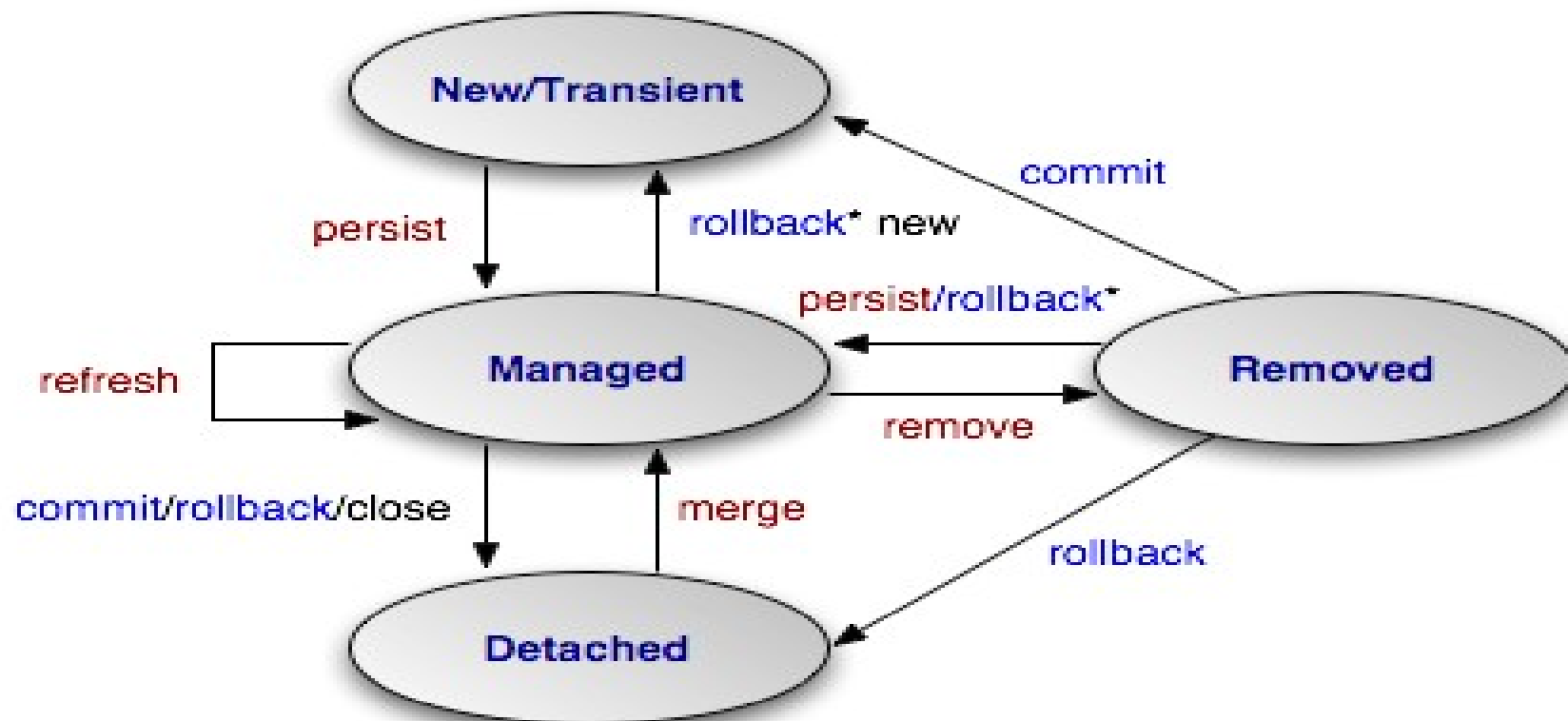
Операции Find и Remove

```
public void removeOrder(Long orderId) {  
    Order order =  
        entityManager.find(Order.class, orderId);  
  
    // Объект будет удален из БД при очередной  
    // операции записи в БД. Доступ к удаленному  
    // объекту приводит к непредсказуемым  
    // результатам.  
    entityManager.remove(order);  
}
```

Операция Merge

```
public OrderLine updateOrderLine(OrderLine orderLine) {  
  
    // Метод merge возвращает управляемую копию  
    // переданного отсоединенного объекта. Если состояние  
    // отсоединенного объекта было изменено, то изменения  
    // будут отражены в возвращаемой копии.  
    return entityManager.merge(orderLine);  
}
```


Жизненный цикл Entity



* = Extended persistence context

Как создать EntityManager

- Разные типы EntityManager создаются по разному
 - > Управляемый контейнером EntityManager (Java EE) создается контейнером и становится доступным для приложения через механизм инъекций. Используется аннотация *@PersistenceContext*.
 - > Управляемый приложением EntityManager (Java SE) создается и закрывается (уничтожается) приложением.

Relationship Mappings – ManyToMany

owner
of Relationship

```
@Entity(access=FIELD)
public class Customer {
    @Id
    int id;
    ...
    @ManyToMany
    Collection<Phone> phones;
}
```

Inverse
of Relationship

```
@Entity(access=FIELD)
public class Phone {
    @Id
    int id;
    ...
    @ManyToMany(mappedBy="phones")
    Collection<Customer> custs;
}
```



Спасибо!

