Mapping Objects With JPA

Java Persistence API 2.0

Aaron Schram

University of Colorado at Boulder

Me

- PhD Candidate at the University of Colorado
- Prior to returning to CU I held several software engineering positions
  - Mocapay, Inc. (Mobile Payments)
  - Rally Software Development (Agile Tooling)
  - BEA Systems (Weblogic Portal, Now Oracle)
  - Lockheed Martin (IS & GS)

Some History

History

- A result of the JSR 317 Expert Group
    - Members included
        - Sun Microsystems, Inc.
        - Oracle
        - BEA Systems*
        - IBM
        - VMWare

History Cont...

- Developed as a replacement for EJB 2 entity beans
- Version 2.0 was released Dec 10th, 2009
- Covers 2 areas of Object Relational Mapping (ORM)
  - Object relational metadata
  - Java Persistence Query Language (JPQL)

History Cont...

- JPA 2.0 included consensus approval for new features
  - Expanded ORM functionality
  - Criteria query API
  - Standardization of query hints
  - Standardization of metadata for DDL generation
  - Validation support

It’s Just A Specification*

- JPA is a specification used to detail what a reference provider should conform to when providing ORM functionality
  - It’s actually more than just a specification
  - A finalized Java Specification Request will include a reference implementation
  - Since JPA is a finalized JSR an implementation is provided
- There are many JPA reference implementations
  - Hibernate, EclipseLink, OpenJPA

Hibernate

- The most popular JPA vendor is Hibernate (JBoss)
- JPA 1.0 was heavily influenced by Gavin King, the creator of Hibernate
  - Much of what exists in JPA is adopted directly from the Hibernate project
  - Many key concepts such as mapping syntax and central session/entity management exist in both

Key Concepts

- JPA utilizes annotated Plain Old Java Objects (POJOs)
  - Define an EntityBean for persistence
    - @Entity
  - Define relationships between beans
    - @OneToOne
    - @OneToMany
    - @ManyToOne
    - @ManyToMany

Key Concepts Cont...

- Primitive types and wrappers are mapped by default
  - String, Long, Integers, Double, etc.
- Mappings can be defined on instance vars or on accessor methods of the POJO
- Supports inheritance and embedding
- EntityManger is used to manage the state and life cycle of all entities within a give persistence context
- Primary keys are generated and accessed via @Id annotation

An Example

Office-Employees Example

- This was a common interview question at one of my previous employers

Question:

How could you model an employee management system using an ORM?

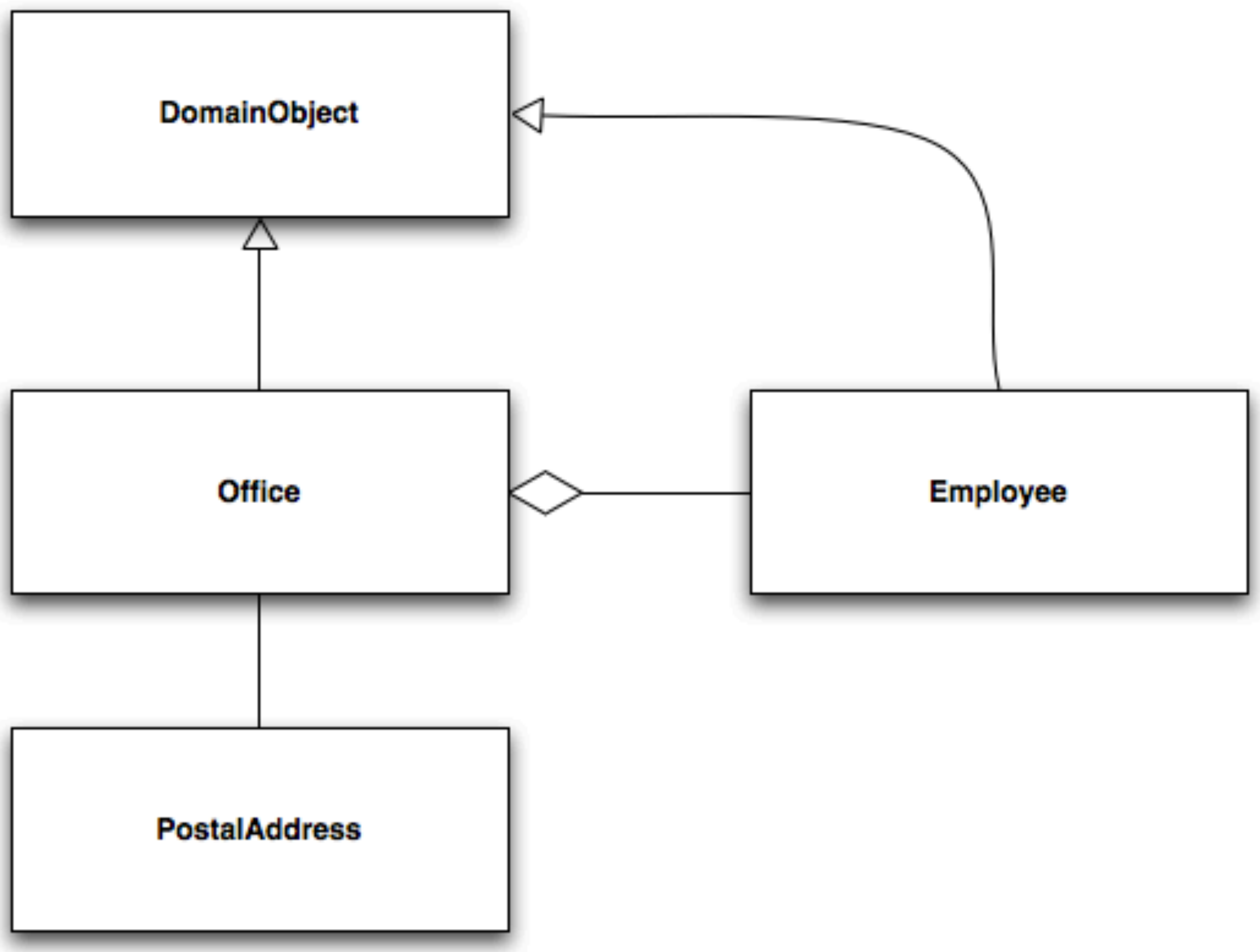
Question Details

In the interview we would build the whole application

- Design an application that allows a customer to view all employees that physically reside in a specific office
- Each employee may only reside in one office
- Employees must have
  - First name, last name, phone number, id
- Each office must have
  - Name, postal address, id
- Any ORM will do, we'll use JPA...

Here, we'll just build out the model tier

The Model


DomainObject
Office
Employee
PostalAddress

From Model to Code

- Our model contains four classes
  - Office
  - Employee
  - DomainObject
  - PostalAddress
- Office and Employee inherit from DomainObject
- DomainObject holds on to best practice attributes such as id, creation date, modified date, version, etc.

From Model to Code Cont...

- @Entity must be used to tell JPA which classes are eligible for persistence
- @ManyToOne must be used to tell JPA there is an aggregation between Office and Employee
- We'll show a use of @Embedded and @Embeddable for the Office-PostalAddress relationship
- As well as inheritance using @MappedSuperclass

DomainObject

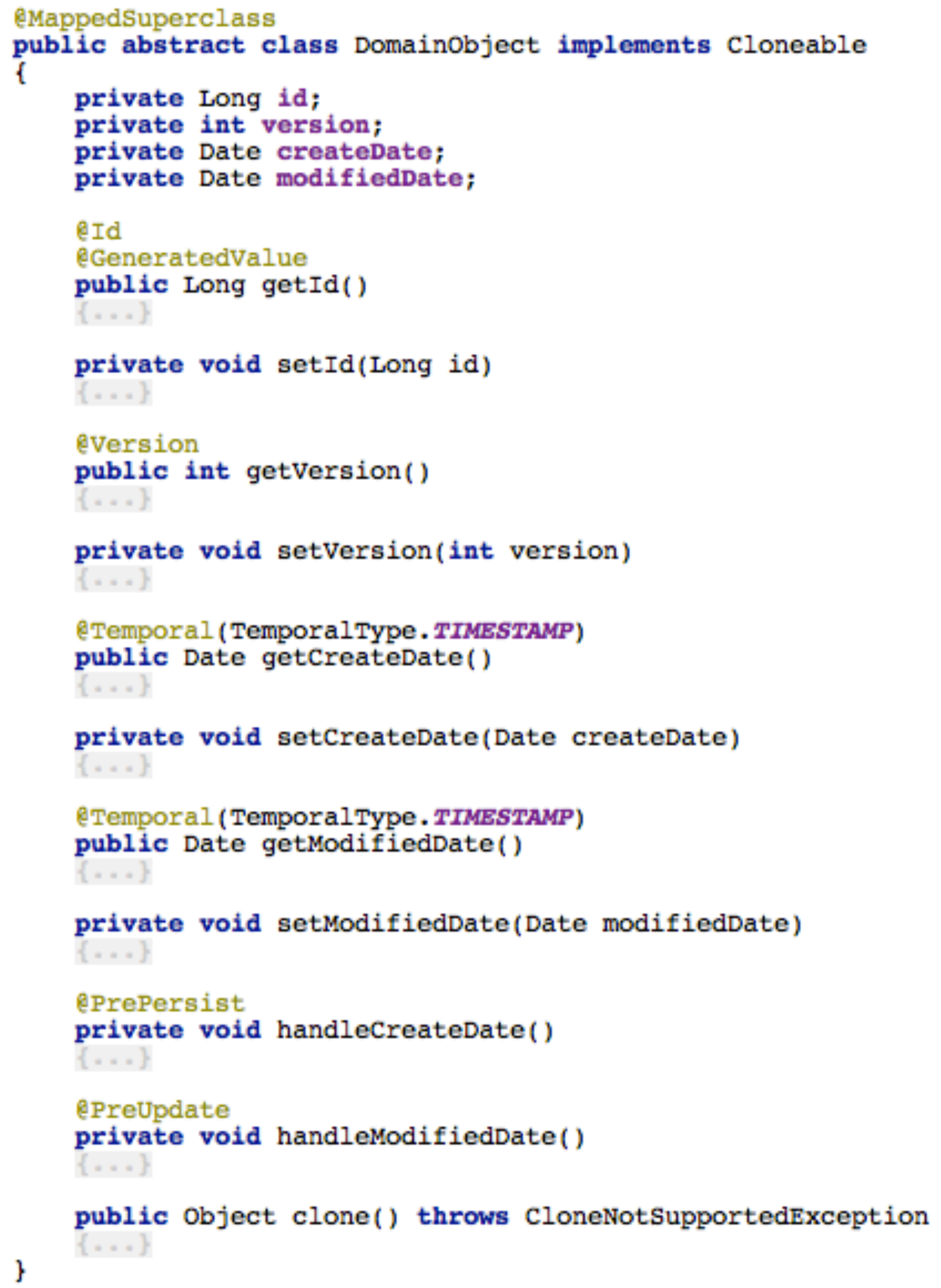
This class is not to be directly persisted

DB generated Id

For optimistic locking

Store as datetime

Call these methods before creation and modification

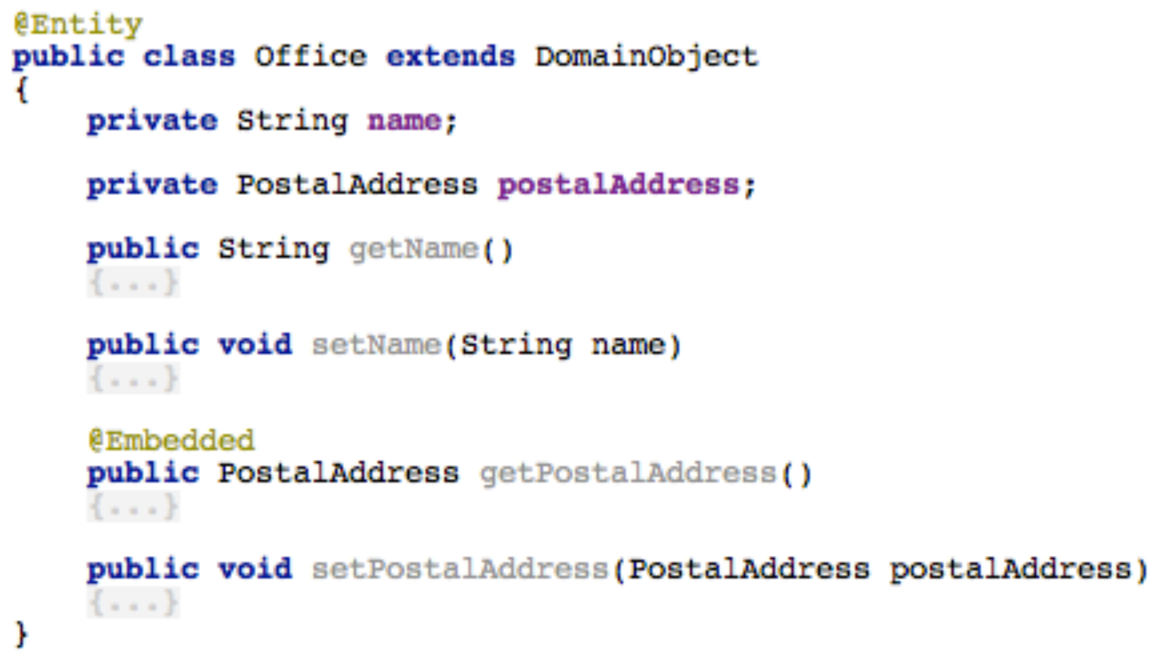


```
@MappedSuperclass
public abstract class DomainObject implements Cloneable
{
    private Long id;
    private int version;
    private Date createDate;
    private Date modifiedDate;

    @Id
    @GeneratedValue
    public Long getId()
    {...}

    private void setId(Long id)
    {...}

    @Version
    public int getVersion()
    {...}

    private void setVersion(int version)
    {...}

    @Temporal(TemporalType.TIMESTAMP)
    public Date getCreateDate()
    {...}

    private void setCreateDate(Date createDate)
    {...}

    @Temporal(TemporalType.TIMESTAMP)
    public Date getModifiedDate()
    {...}

    private void setModifiedDate(Date modifiedDate)
    {...}

    @PrePersist
    private void handleCreateDate()
    {...}

    @PreUpdate
    private void handleModifiedDate()
    {...}

    public Object clone() throws CloneNotSupportedException
    {...}
}
```

Office

Eligible for persistence

Embed PostalAddress in the same table as Office

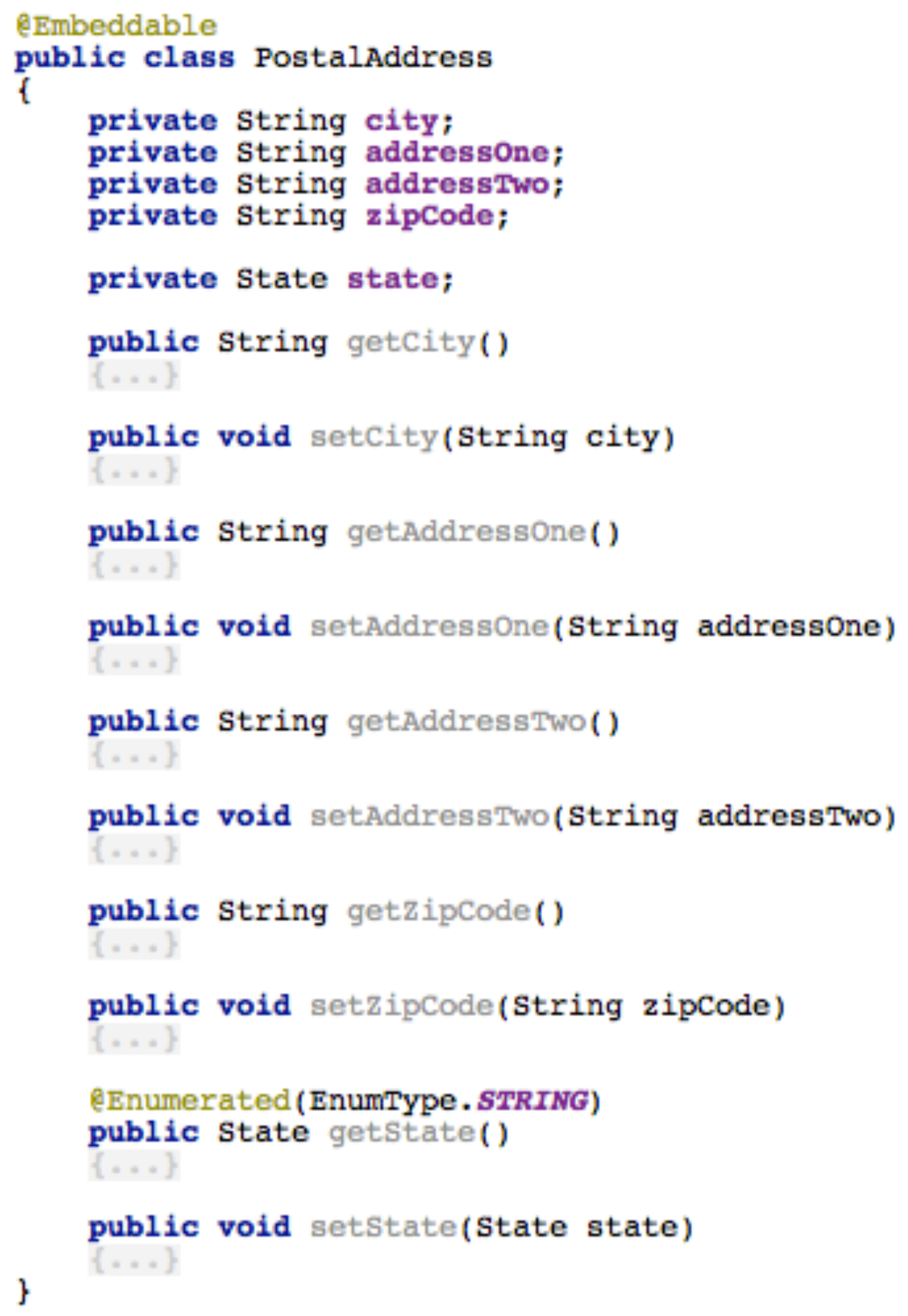


```
@Entity
public class Office extends DomainObject
{
    private String name;

    private PostalAddress postalAddress;

    public String getName()
    {...}

    public void setName(String name)
    {...}

    @Embedded
    public PostalAddress getPostalAddress()
    {...}

    public void setPostalAddress(PostalAddress postalAddress)
    {...}
}
```

PostalAddress

Allow this object to be embedded by other objects

State is an Enum that will be treated as a String (varchar)

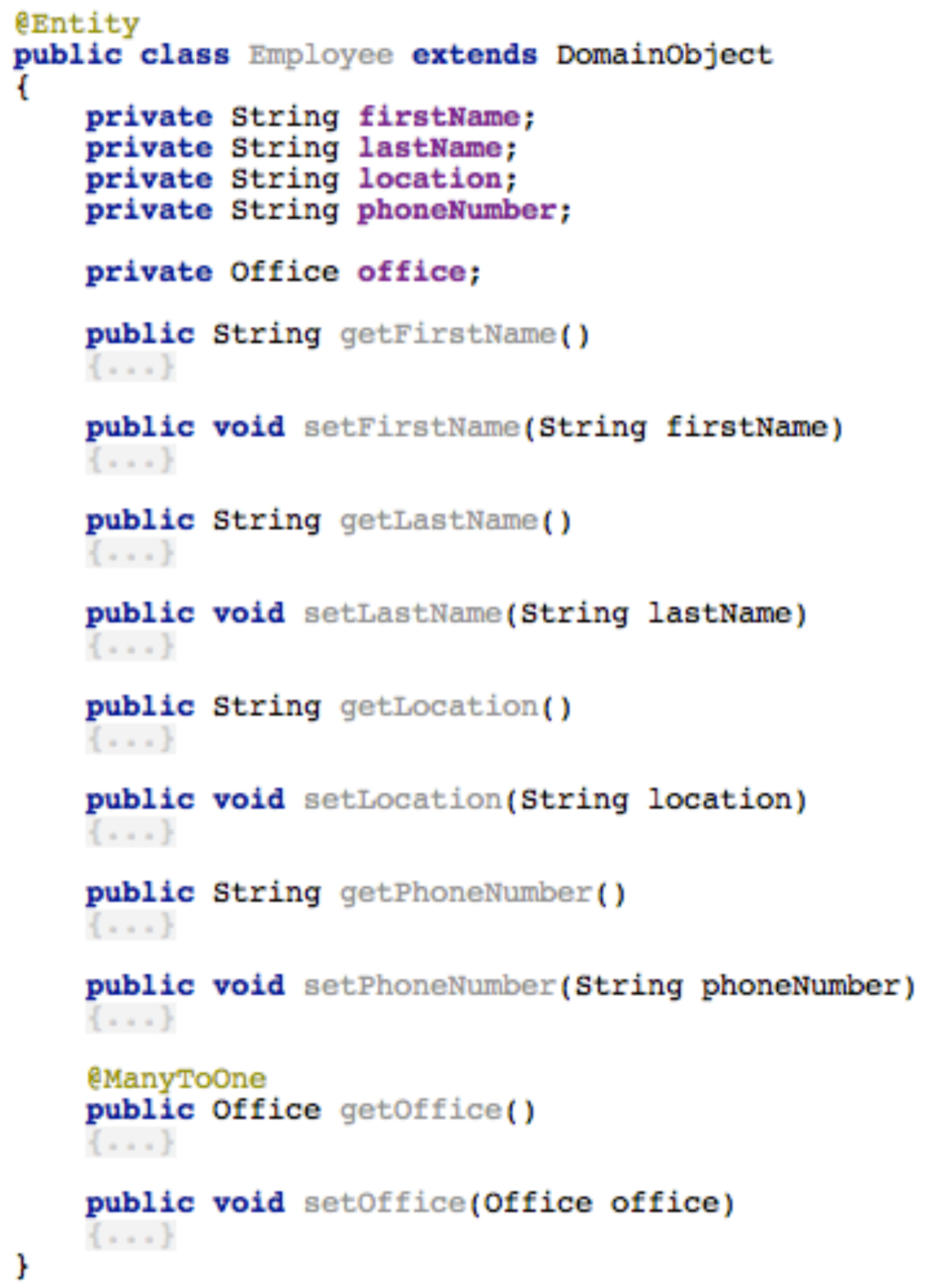


```
@Embeddable
public class PostalAddress
{
    private String city;
    private String addressOne;
    private String addressTwo;
    private String zipCode;

    private State state;

    public String getCity()
    {...}

    public void setCity(String city)
    {...}

    public String getAddressOne()
    {...}

    public void setAddressOne(String addressOne)
    {...}

    public String getAddressTwo()
    {...}

    public void setAddressTwo(String addressTwo)
    {...}

    public String getZipCode()
    {...}

    public void setZipCode(String zipCode)
    {...}

    @Enumerated(EnumType.STRING)
    public State getState()
    {...}

    public void setState(State state)
    {...}
}
```

Employee

Eligible for persistence

Defines the many to one association with Office



```
@Entity
public class Employee extends DomainObject
{
    private String firstName;
    private String lastName;
    private String location;
    private String phoneNumber;

    private Office office;

    public String getFirstName()
    {...}

    public void setFirstName(String firstName)
    {...}

    public String getLastName()
    {...}

    public void setLastName(String lastName)
    {...}

    public String getLocation()
    {...}

    public void setLocation(String location)
    {...}

    public String getPhoneNumber()
    {...}

    public void setPhoneNumber(String phoneNumber)
    {...}

    @ManyToOne
    public Office getOffice()
    {...}

    public void setOffice(Office office)
    {...}
}
```

Explanation

- @Embeddable and @Embedded
    - Allows for the attributes of an embedded class to be stored in the same table as the embedding class
- @Enumerated
    - Allows for the value of an Enum to be stored in a column in the class's database table
- @MappedSuperclass
    - Allows for all attributes of the superclass to be utilized by the subclasses
    - Duplicates all superclass attributes on subclass tables

The Database

The Database

- JPA is capable of generating the underlying database for the developer
- Most aspects of the generation are available for customization
  - The defaults are generally good enough
- Any @Entity causes the generation of a database table. Our generated tables are:
    - Office table
    - Employee table

Office Table

Field	Type
id	bigint(20)
createDate	datetime
modifiedDate	datetime
version	int(11)
name	varchar(255)
addressOne	varchar(255)
addressTwo	varchar(255)
city	varchar(255)
state	varchar(255)
zipCode	varchar(255)

Employee Table

Field	Type
id	bigint(20)
createDate	datetime
modifiedDate	datetime
version	int(11)
firstName	varchar(255)
lastName	varchar(255)
location	varchar(255)
phoneNumber	varchar(255)
office_id	bigint(20)

FK to Office

Take Aways

Take Aways

- JPA is a specification that a developer can code to in order to easily leverage ORM technologies
- There are a wide variety of vendors that implement the specification
    - Coding to the spec allows the developer to be flexible in their choice of vendor implementations with limited ripple throughout the codebase
- JPA greatly simplifies persistence of POJOs through a small set of easily utilized annotations

Questions?

Aaron Schram

aaron.schram@colorado.edu