Java Persistence API (JPA) Step by Step

Albert Guo

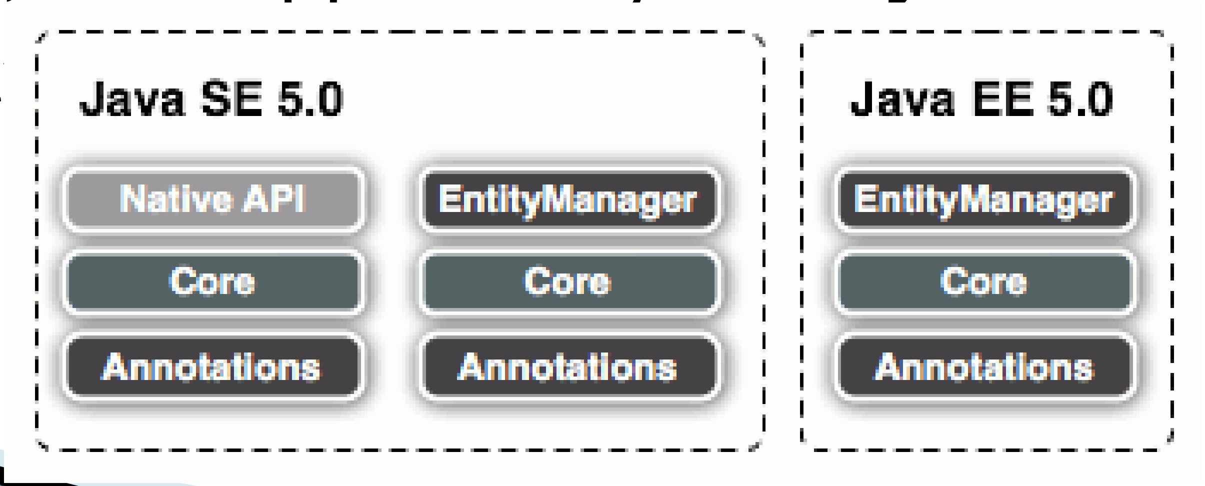
junyuo@gmail.com

Agenda

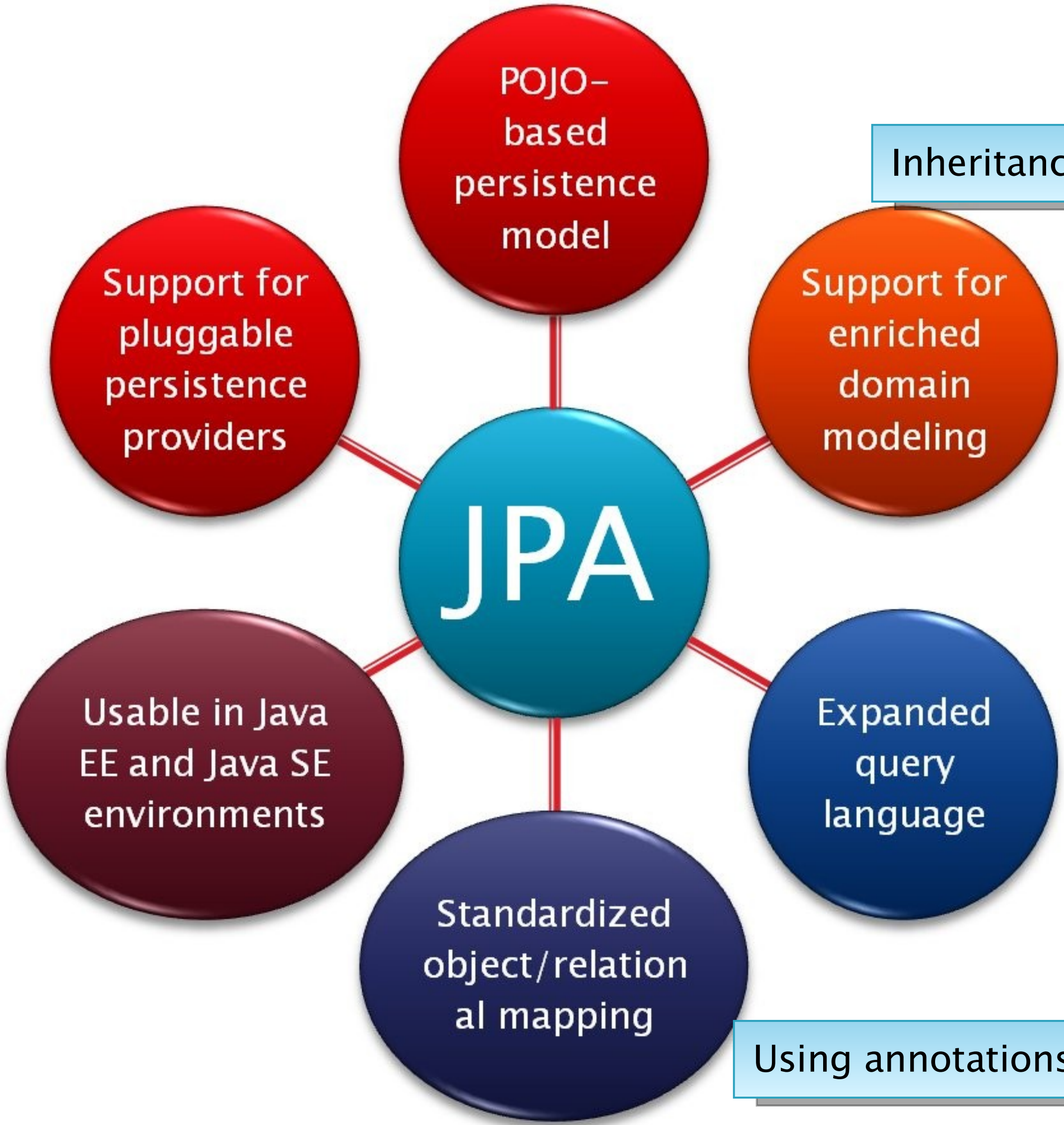
- What is Java Persistence API
- Primary Features
- Five Steps to Implement JPA
  - Download Hibernate Components
  - Prepare Database, and Download JDBC Driver
  - Implemented POJO entities and add annotations
  - Persistence.xml
  - Implemented client side code via EntityManager

What is Java Persistence API

- The Java Persistence API is the standard object/relational mapping and persistence management interface of the Java EE 5.0 platform. As part of the EJB 3.0 specification effort, it is supported by all major vendors of the Ja



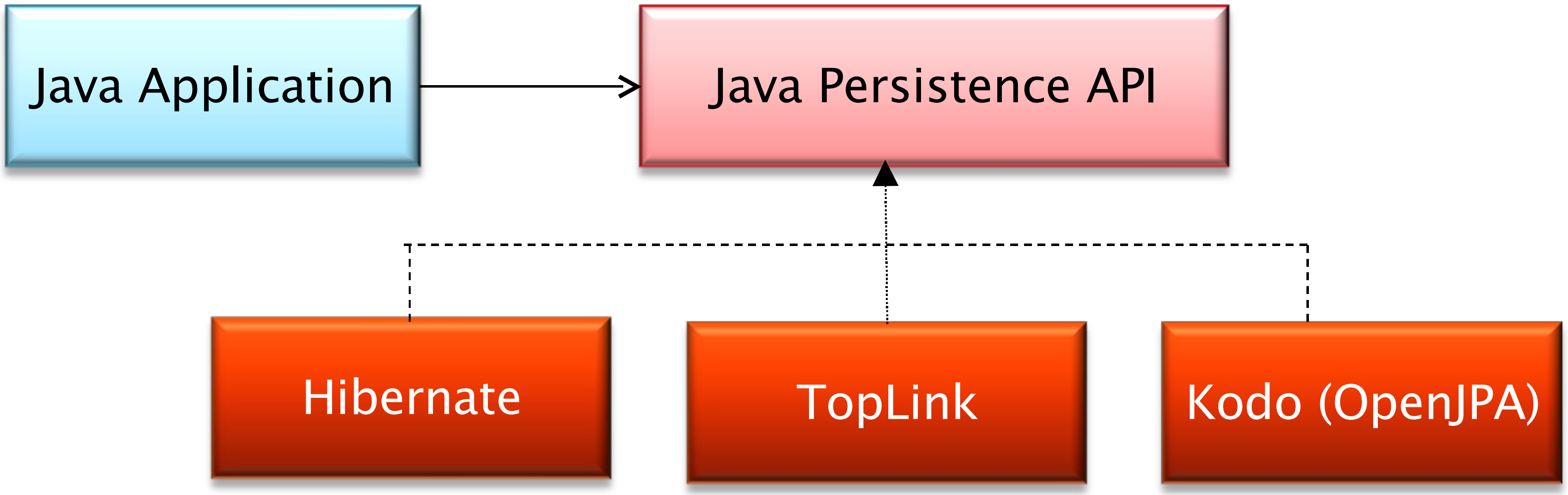
Primary Features


POJO-based persistence model
Support for pluggable persistence providers
Support for enriched domain modeling
JPA
Usable in Java EE and Java SE environments
Expanded query language
Standardized object/relational mapping

Inheritance, polymorphism, etc.

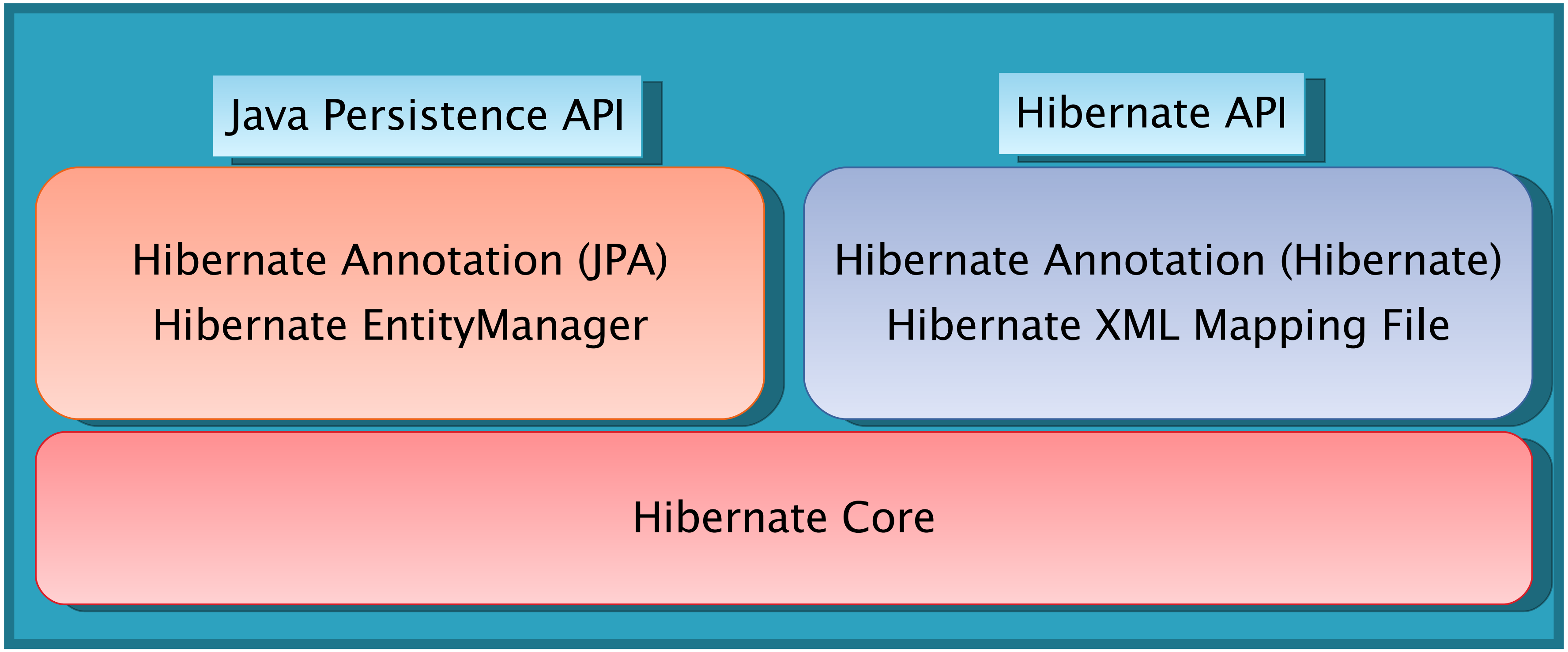
Using annotations and/or XML

JPA Architecture

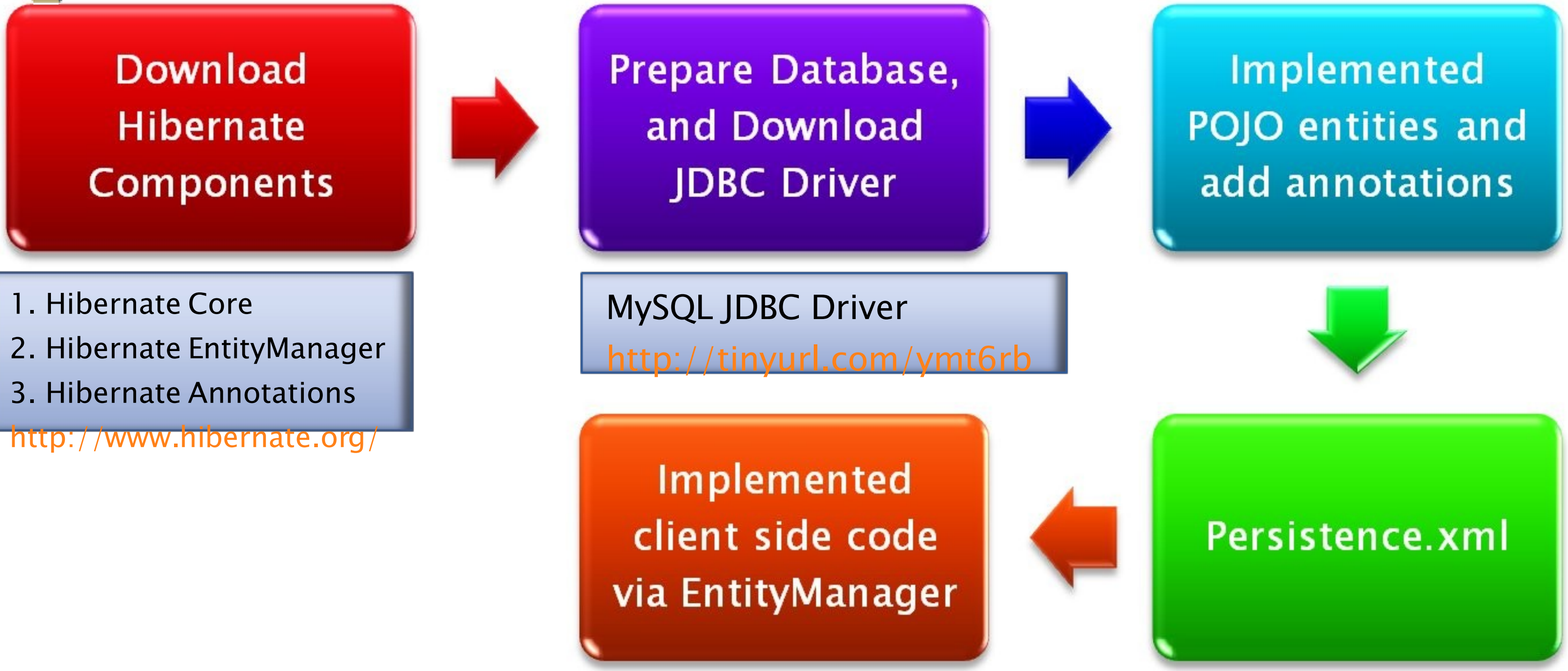


Everyone can use their own favorite persistence technology

JPA and Hibernate


Java Persistence API
Hibernate API
Hibernate Annotation (JPA)
Hibernate EntityManager
Hibernate Annotation (Hibernate)
Hibernate XML Mapping File
Hibernate Core

Five Steps to Implement JPA

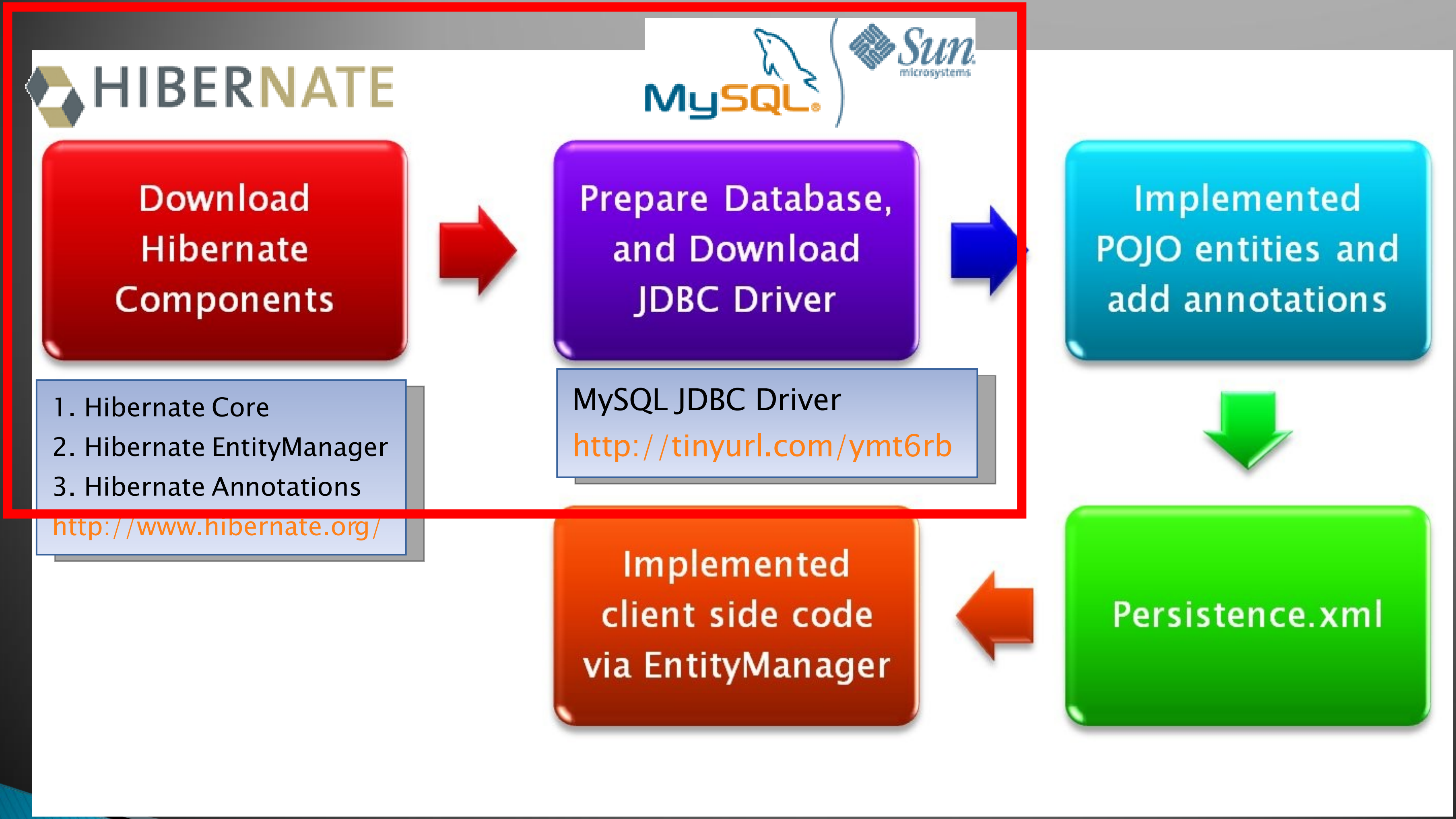

Download Hibernate Components
Prepare Database, and Download JDBC Driver
Implemented POJO entities and add annotations
Persistence.xml
Implemented client side code via EntityManager

1. Hibernate Core
2. Hibernate EntityManager
3. Hibernate Annotations

http://www.hibernate.org/

MySQL JDBC Driver
http://tinyurl.com/ymt6rb

Download Hibernate Components


HIBERNATE
MySQL
Sun microsystems
Download Hibernate Components
Prepare Database, and Download JDBC Driver
Implemented POJO entities and add annotations
Persistence.xml
Implemented client side code via EntityManager

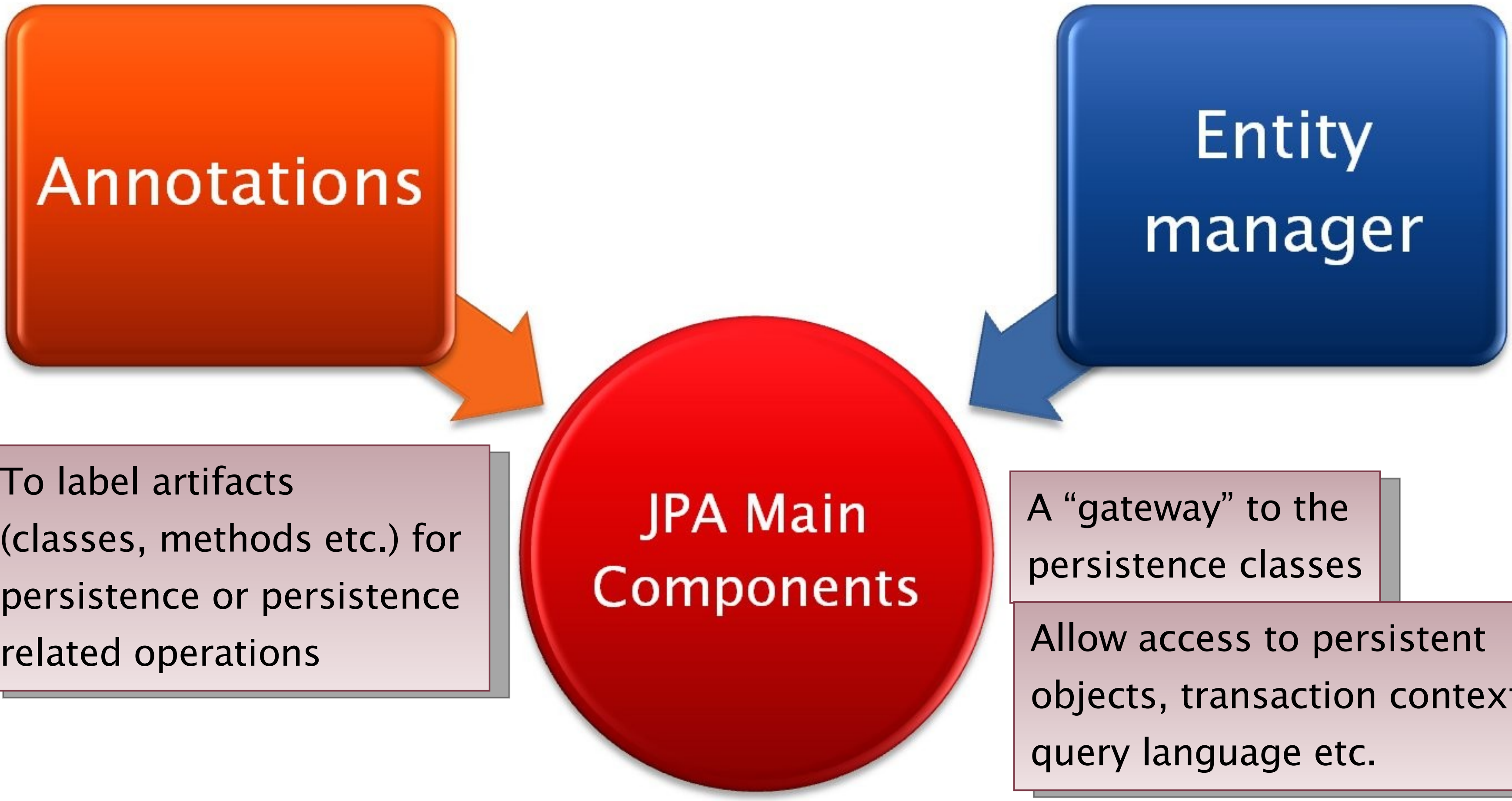
1. Hibernate Core
2. Hibernate EntityManager
3. Hibernate Annotations

http://www.hibernate.org/

MySQL JDBC Driver

http://tinyurl.com/ymt6rb

JPA Main Components



To label artifacts (classes, methods etc.) for persistence or persistence related operations

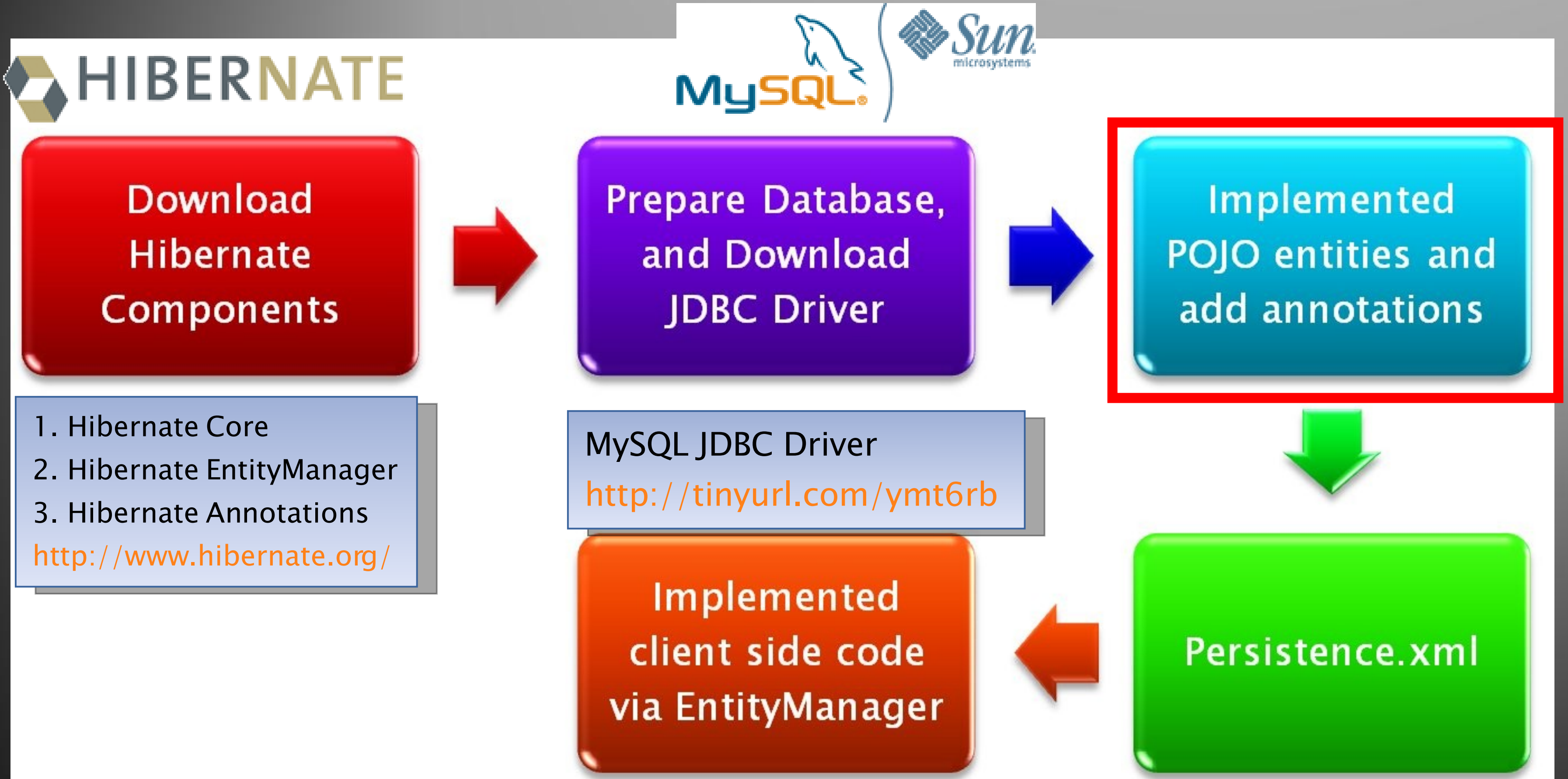
A “gateway” to the persistence classes

Allow access to persistent objects, transaction context, query language etc.

Hibernate Annotations / EntityManager

- Hibernate Annotations includes
  - Standardized Java Persistence and EJB 3.0 (JSR 220) object/relational mapping annotations
  - Hibernate-specific extension annotations for performance optimization and special mappings
- Hibernate EntityManager includes
  - The standard Java Persistence management API
  - The standard Java Persistence Query Language
  - The standard Java Persistence object lifecycle rules
  - The standard Java Persistence configuration and packaging

Download Hibernate Components

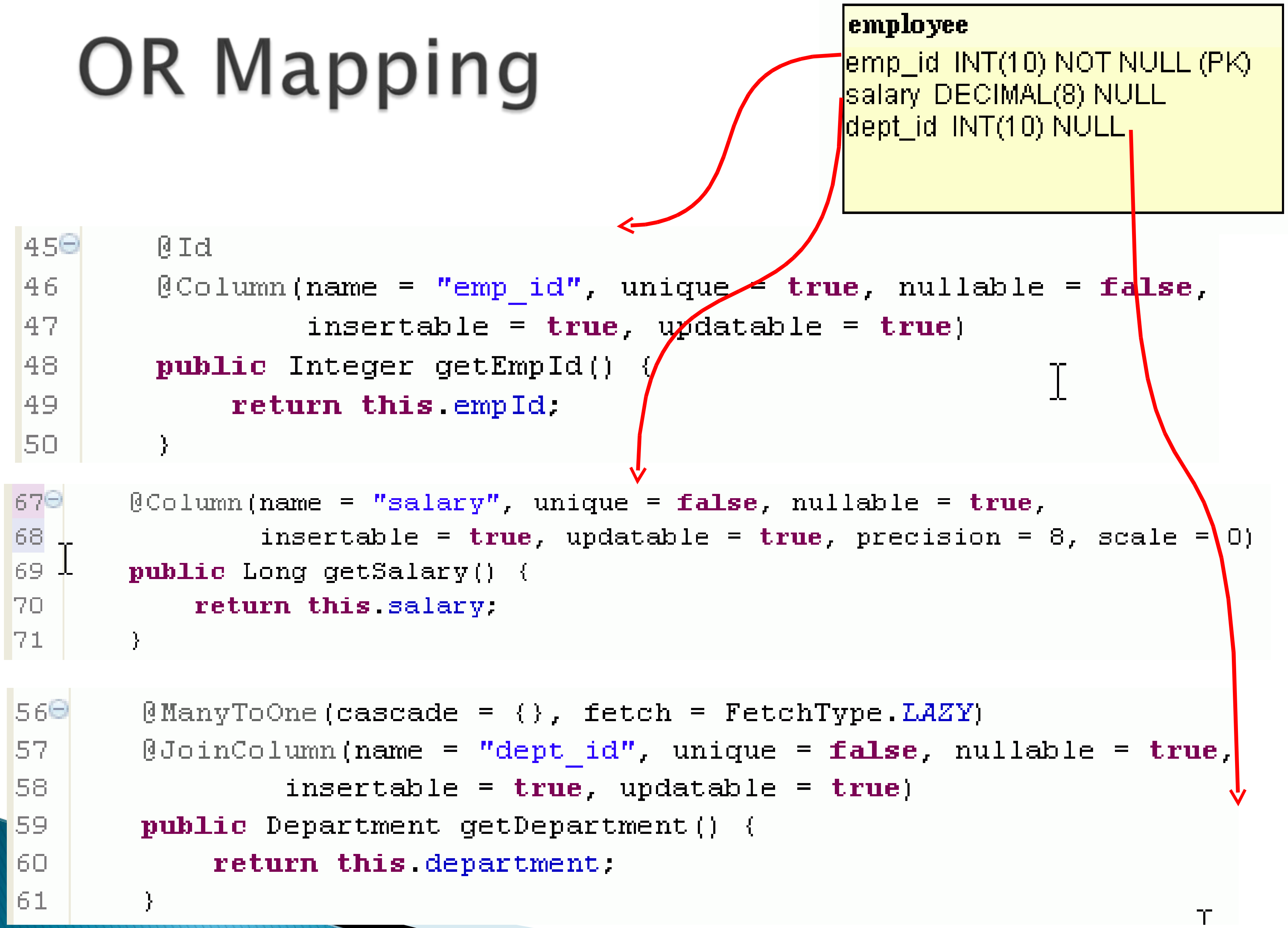

HIBERNATE
MySQL
Sun microsystems
Download Hibernate Components
Prepare Database, and Download JDBC Driver
Implemented POJO entities and add annotations
Persistence.xml
Implemented client side code via EntityManager

1. Hibernate Core
2. Hibernate EntityManager
3. Hibernate Annotations

http://www.hibernate.org/

MySQL JDBC Driver
http://tinyurl.com/ymt6rb

OR Mapping



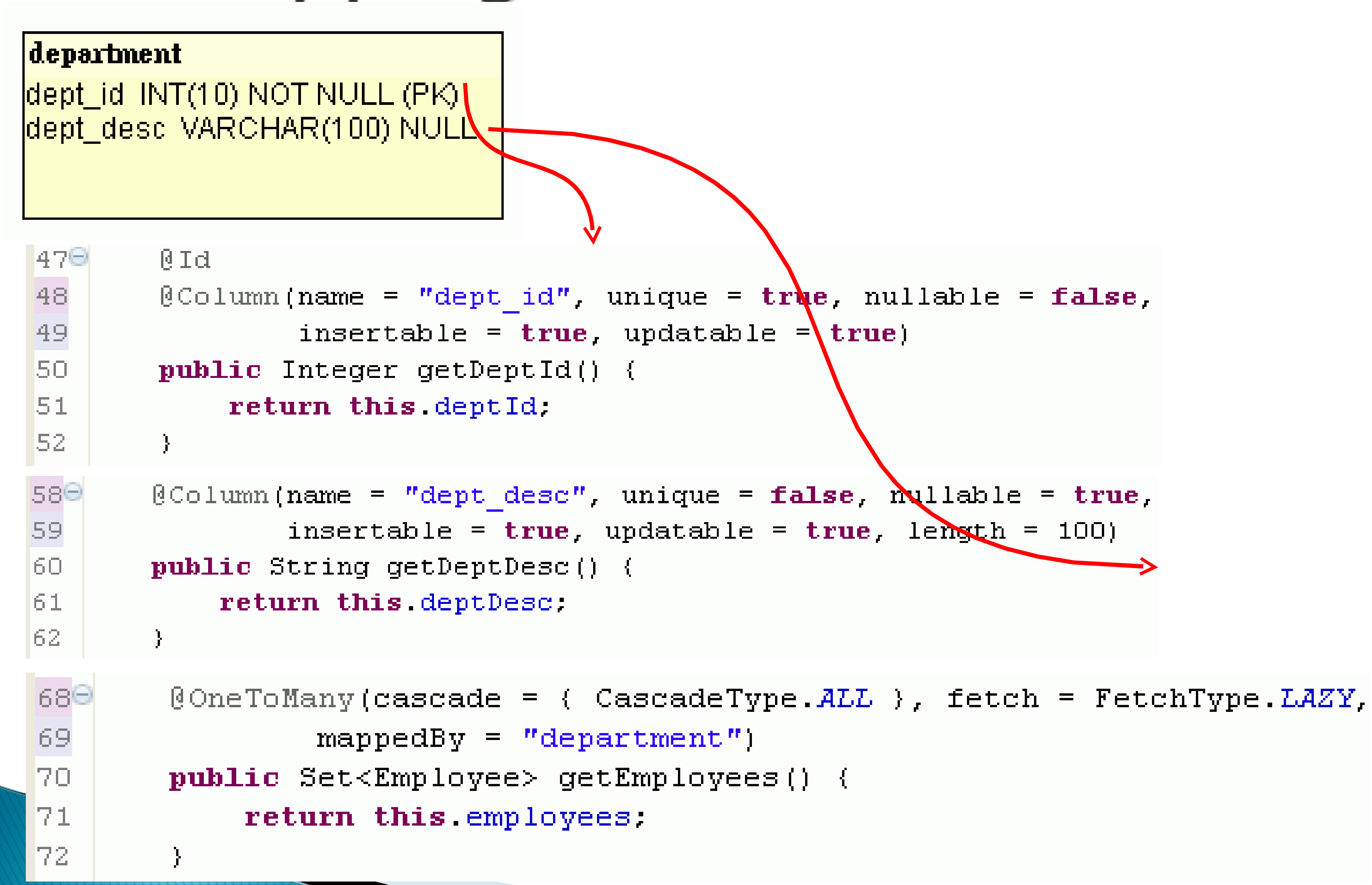
```
employee
emp_id  INT(10) NOT NULL (PK)
salary  DECIMAL(8) NULL
dept_id  INT(10) NULL
```

```java
    @Id
    @Column(name = "emp_id", unique = true, nullable = false,
            insertable = true, updatable = true)
    public Integer getEmpId() {
        return this.empId;
    }
```

```java
    @Column(name = "salary", unique = false, nullable = true,
            insertable = true, updatable = true, precision = 8, scale = 0)
    public Long getSalary() {
        return this.salary;
    }
```

```java
    @ManyToOne(cascade = {}, fetch = FetchType.LAZY)
    @JoinColumn(name = "dept_id", unique = false, nullable = true,
            insertable = true, updatable = true)
    public Department getDepartment() {
        return this.department;
    }
```

OR Mapping



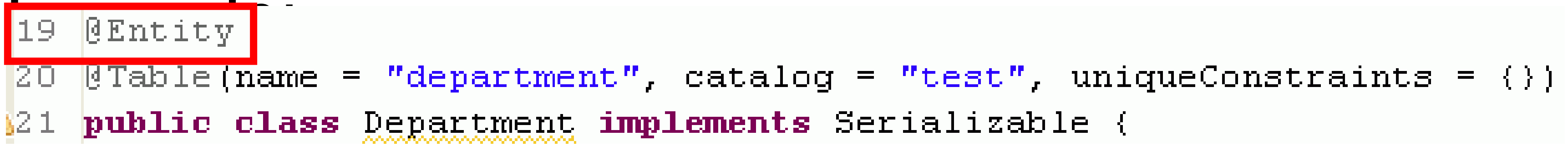
department
dept_id INT(10) NOT NULL (PK)
dept_desc VARCHAR(100) NULL

```
    @Id
    @Column(name = "dept_id", unique = true, nullable = false,
            insertable = true, updatable = true)
    public Integer getDeptId() {
        return this.deptId;
    }
```

```
    @Column(name = "dept_desc", unique = false, nullable = true,
            insertable = true, updatable = true, length = 100)
    public String getDeptDesc() {
        return this.deptDesc;
    }
```

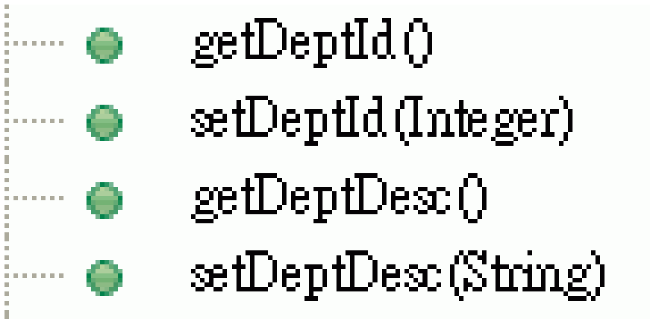
```
    @OneToMany(cascade = { CascadeType.ALL }, fetch = FetchType.LAZY,
            mappedBy = "department")
    public Set<Employee> getEmployees() {
        return this.employees;
    }
```

@Entity

- Attached to a class
- Signify that a class is persistent
- 

```
@Entity
@Table(name = "department", catalog = "test", uniqueConstraints = {})
public class Department implements Serializable {
```

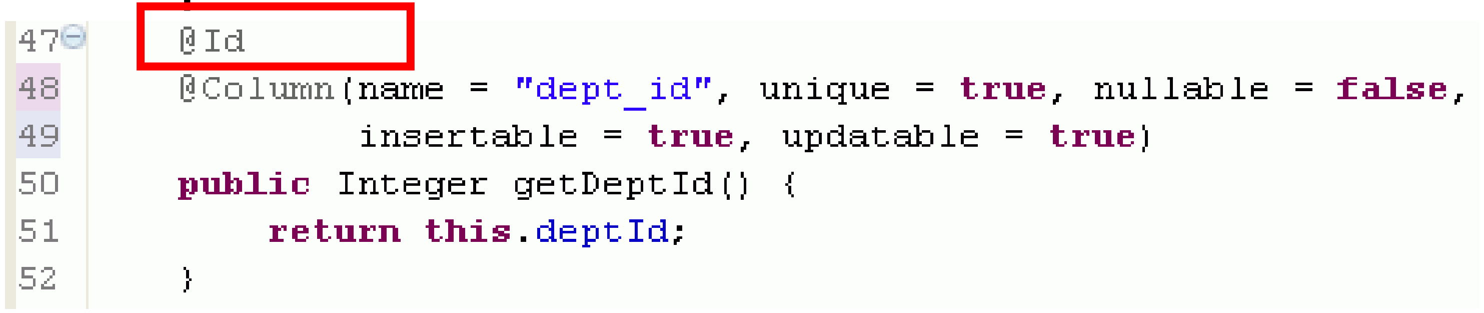
- An entity must follow the Java Bean convention for its attributes to be persistent
  - Having getters and setters



```
getDeptId()
setDeptId(Integer)
getDeptDesc()
setDeptDesc(String)
```

@Id

- Each entity must have an identity
- An identity of an entity could simply be a class variable annotated with @Id
- Example



```
    @Id
    @Column(name = "dept_id", unique = true, nullable = false,
            insertable = true, updatable = true)
    public Integer getDeptId() {
        return this.deptId;
    }
```

@Id

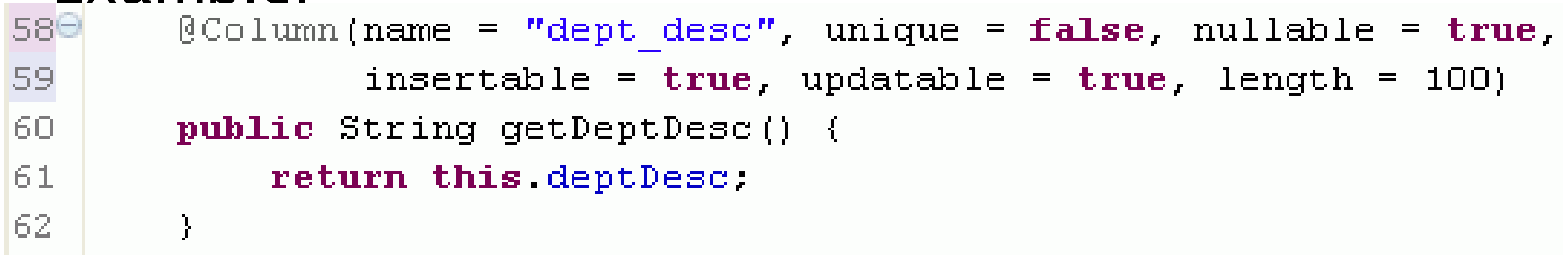
- Id can be auto generated

  @Id(generate=GeneratorType.AUTO)

- There are other strategies such as
  - GeneratorType.SEQUENCE
  - GeneratorType.IDENTITY
- AUTO is best for portability between database vendors

@Column

- @Column, is put on getter of a class variable
- Has several functionalities
  - Updatable (boolean)
  - Nullable (updatable)
  - Length (int)
  - Example:



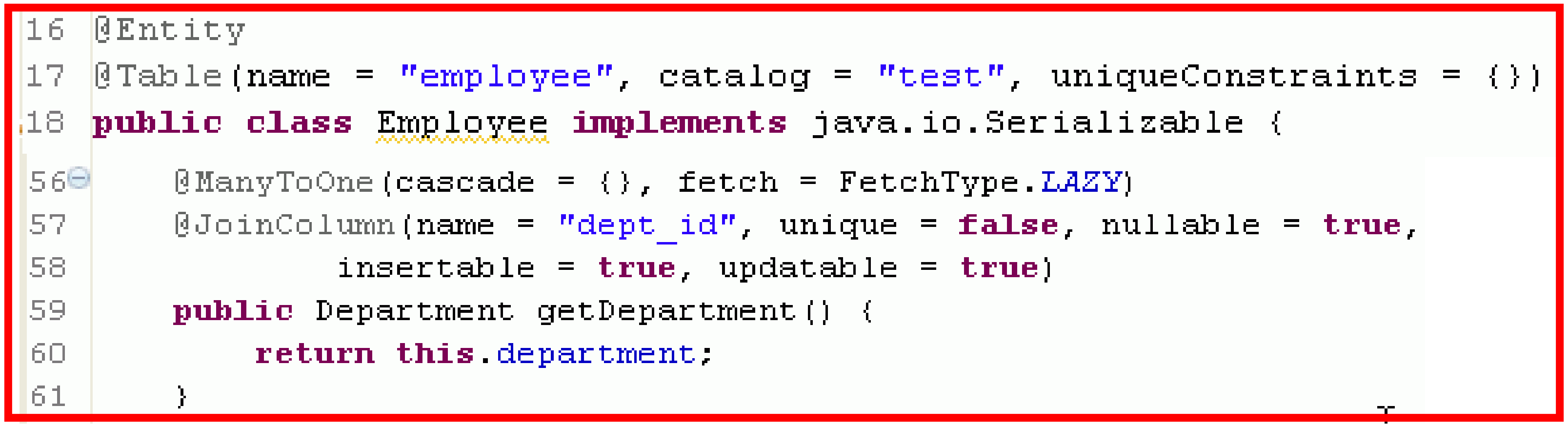
```
    @Column(name = "dept_desc", unique = false, nullable = true,
            insertable = true, updatable = true, length = 100)
    public String getDeptDesc() {
        return this.deptDesc;
    }
```

Linking objects

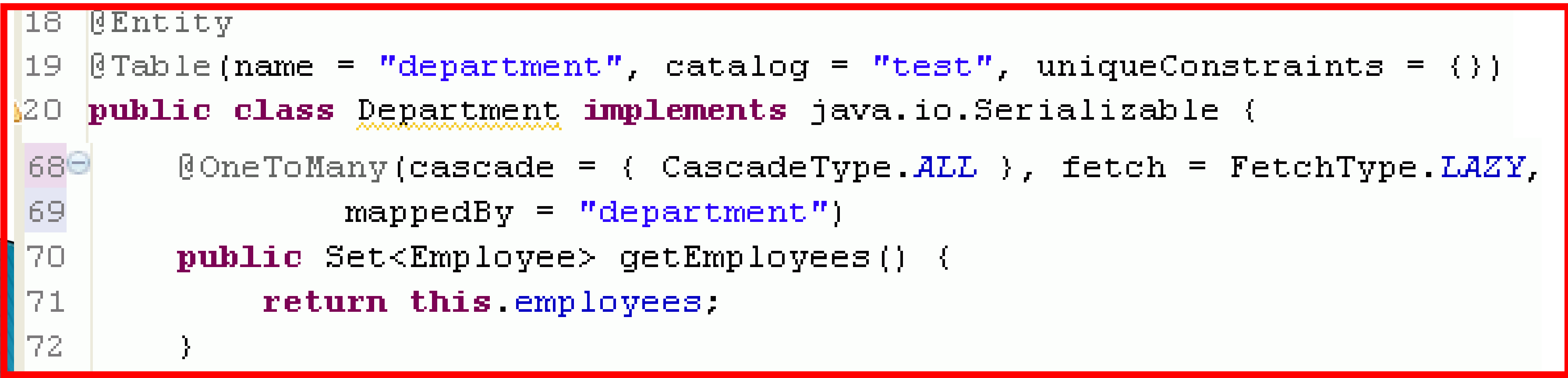
- There are 4 types of links
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- In most cases, putting the annotation on a getter of a class variable would be enough
- In some cases, we need to identify a few parameters to the annotations

@OneToMany, @ManyToOne

- Two cases
  - Two entities share the same primary key value



```java
@Entity
@Table(name = "employee", catalog = "test", uniqueConstraints = {})
public class Employee implements java.io.Serializable {
    @ManyToOne(cascade = {}, fetch = FetchType.LAZY)
    @JoinColumn(name = "dept_id", unique = false, nullable = true,
            insertable = true, updatable = true)
    public Department getDepartment() {
        return this.department;
    }
```



```java
@Entity
@Table(name = "department", catalog = "test", uniqueConstraints = {})
public class Department implements java.io.Serializable {
    @OneToMany(cascade = { CascadeType.ALL }, fetch = FetchType.LAZY,
            mappedBy = "department")
    public Set<Employee> getEmployees() {
        return this.employees;
    }
```

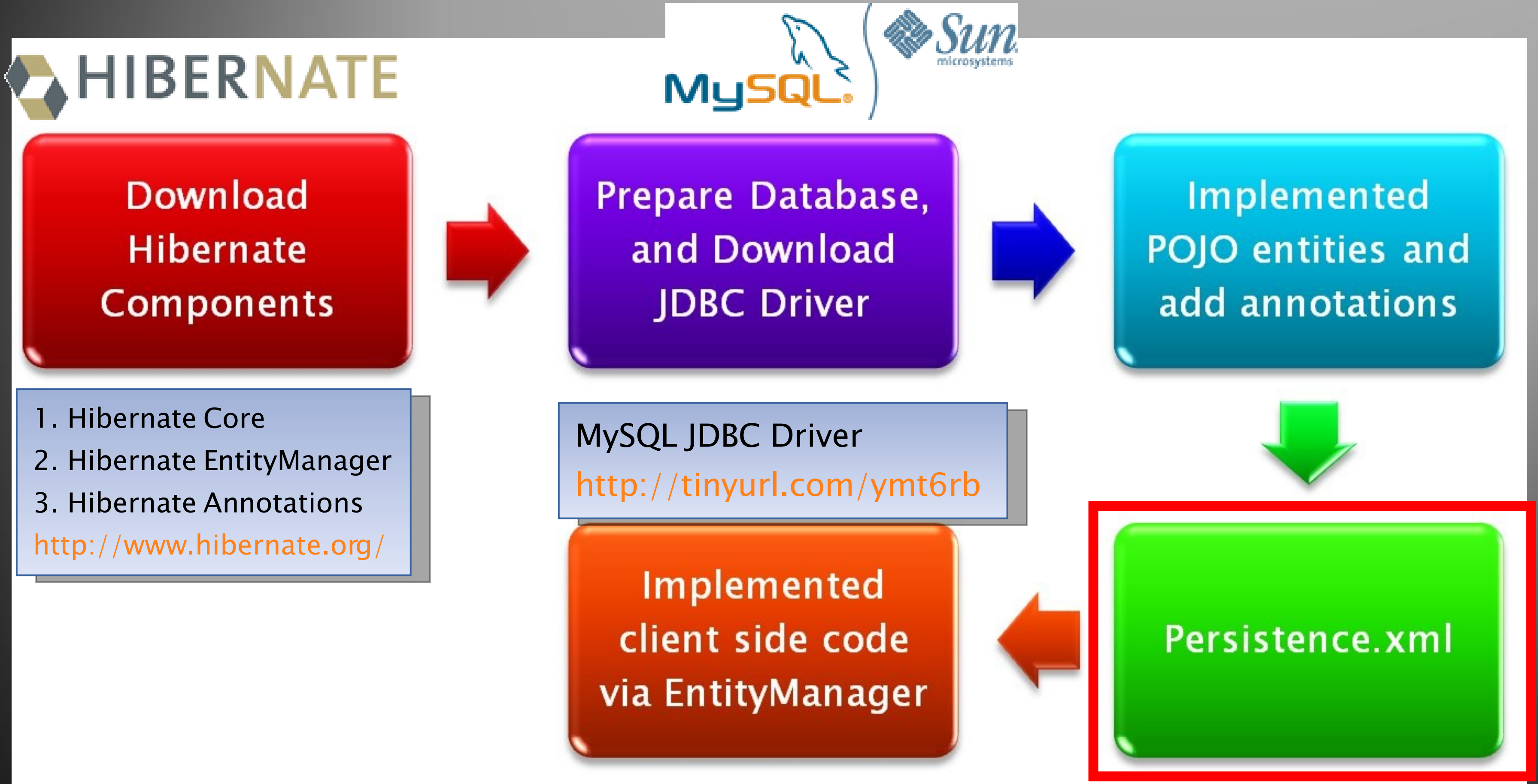
Entity as DTO

- Entity objects have two distinct modes
  - Attached
    - The object is in the database
  - Detached
    - The object is in memory acting as a DTO
    - Modification on detached object would not be persisted automatically
    - Developers need to persist detached objects using a primitive

Entity manager

- Entity manager:
  - Gateway to persistent classes
  - Enable queries
  - Outside of session beans, provides transaction facility

Configure Persistence.xml


HIBERNATE
MySQL
Sun microsystems
Download Hibernate Components
Prepare Database, and Download JDBC Driver
Implemented POJO entities and add annotations
Persistence.xml
Implemented client side code via EntityManager

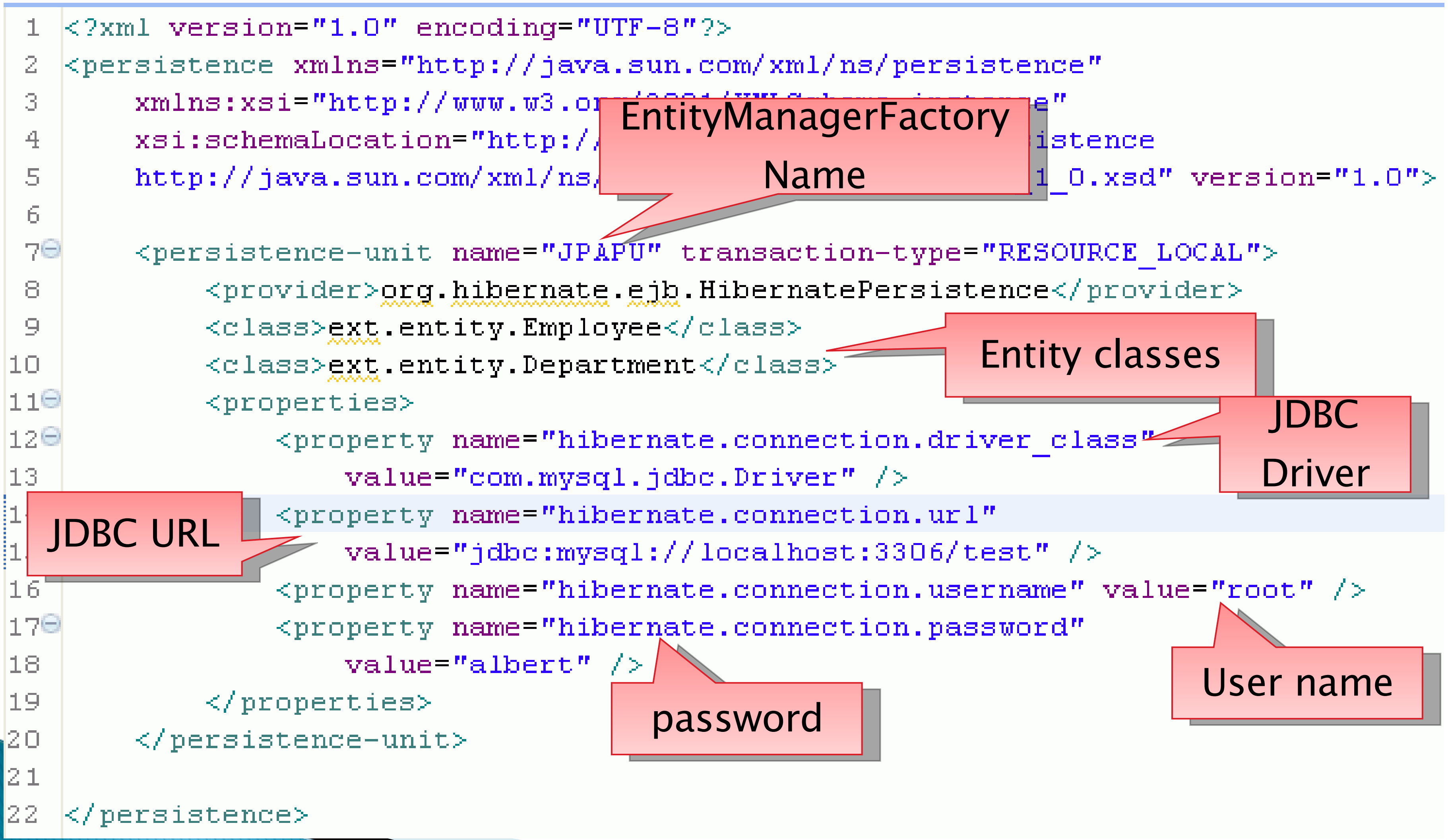
1. Hibernate Core
2. Hibernate EntityManager
3. Hibernate Annotations

http://www.hibernate.org/

MySQL JDBC Driver

http://tinyurl.com/ymt6rb

Configure Persistence.xml

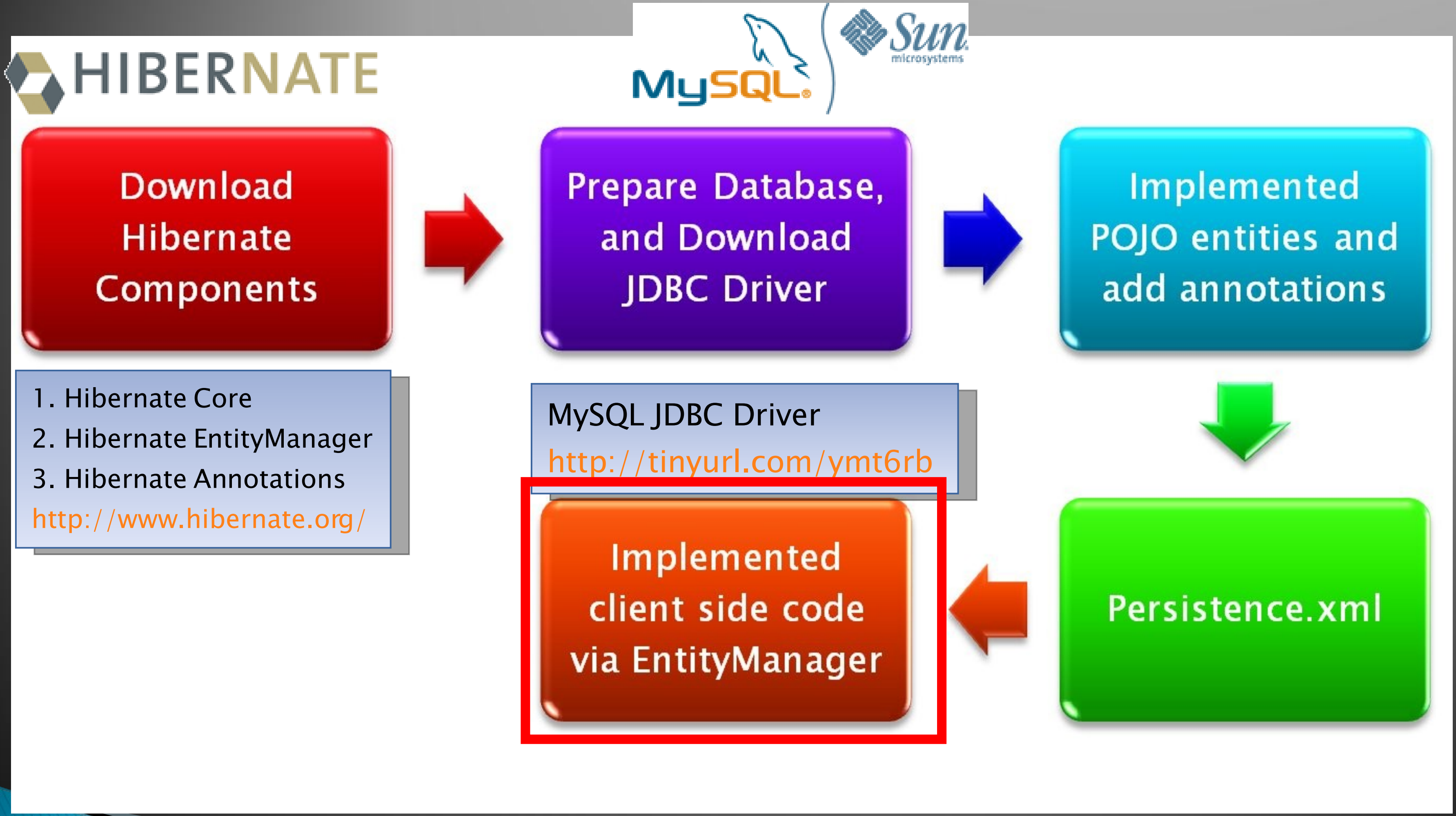


```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.or[illegible]e"
    xsi:schemaLocation="http:/[illegible]istence
    http://java.sun.com/xml/ns/[illegible]1_0.xsd" version="1.0">

    <persistence-unit name="JPAPU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <class>ext.entity.Employee</class>
        <class>ext.entity.Department</class>
        <properties>
            <property name="hibernate.connection.driver_class"
                value="com.mysql.jdbc.Driver" />
            <property name="hibernate.connection.url"
                value="jdbc:mysql://localhost:3306/test" />
            <property name="hibernate.connection.username" value="root" />
            <property name="hibernate.connection.password"
                value="albert" />
        </properties>
    </persistence-unit>

</persistence>
```

Write Client Code


HIBERNATE
MySQL
Sun microsystems
Download Hibernate Components
Prepare Database, and Download JDBC Driver
Implemented POJO entities and add annotations
Persistence.xml
Implemented client side code via EntityManager

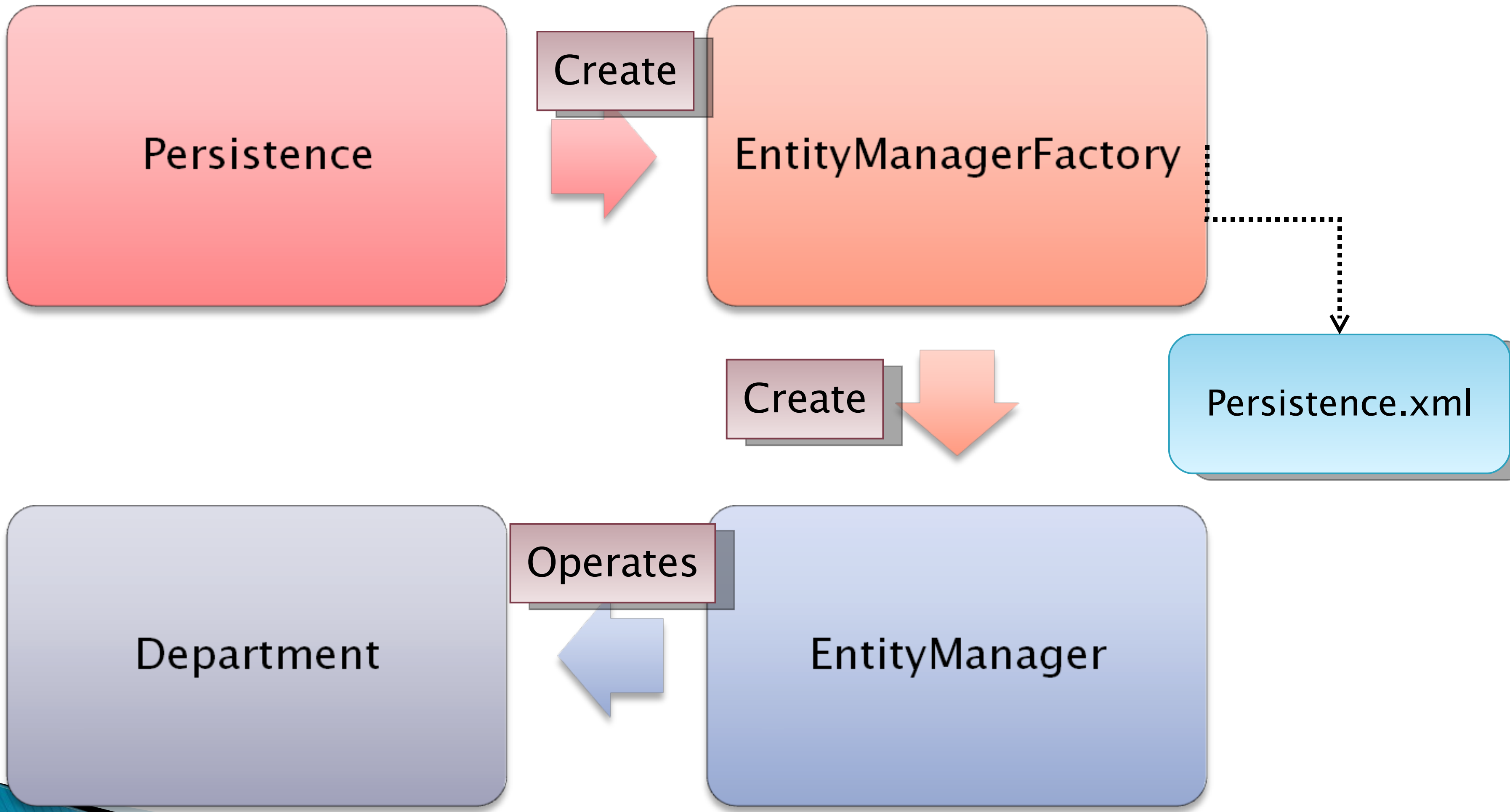
1. Hibernate Core
2. Hibernate EntityManager
3. Hibernate Annotations

http://www.hibernate.org/

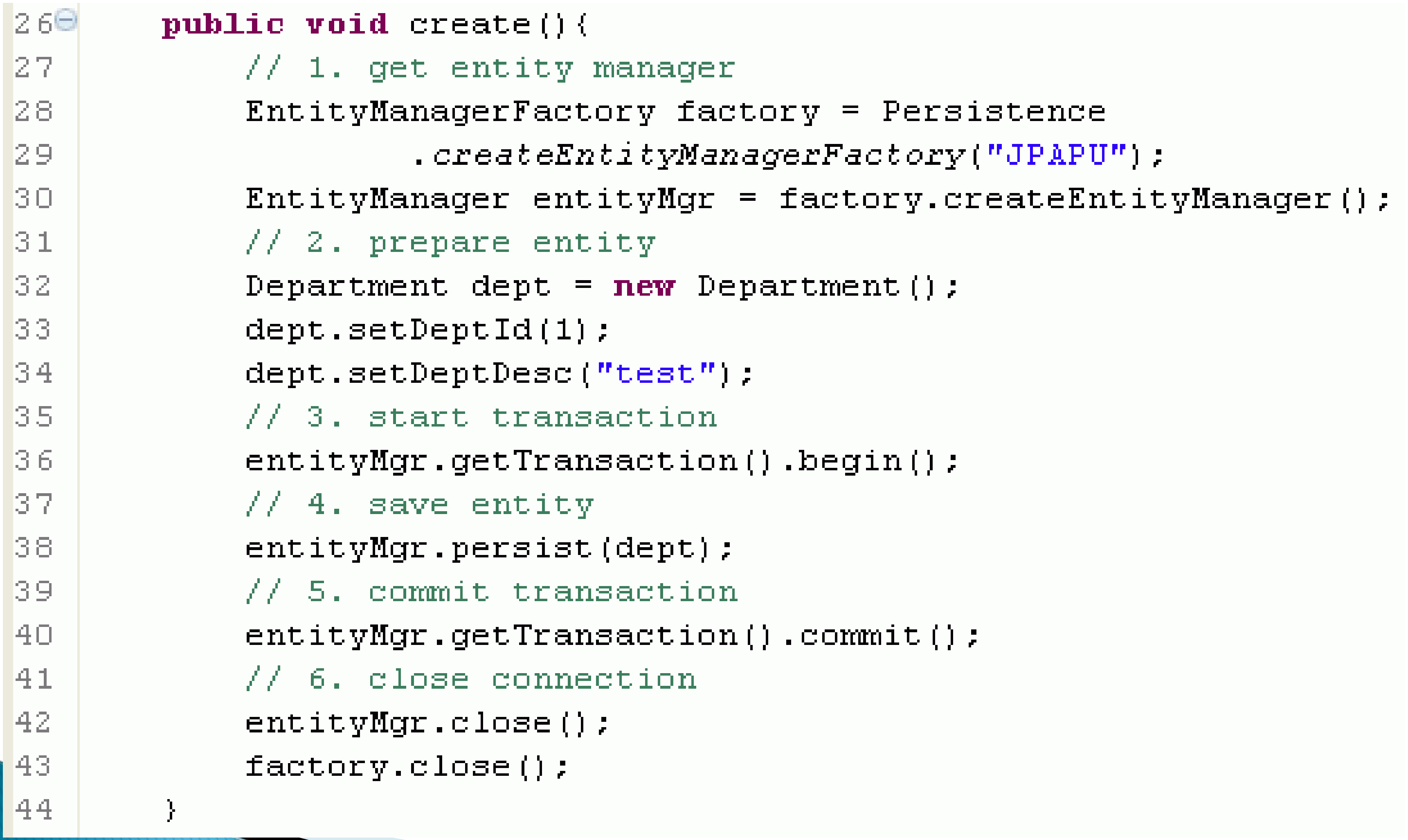
MySQL JDBC Driver

http://tinyurl.com/ymt6rb

JPA Data Operation Process

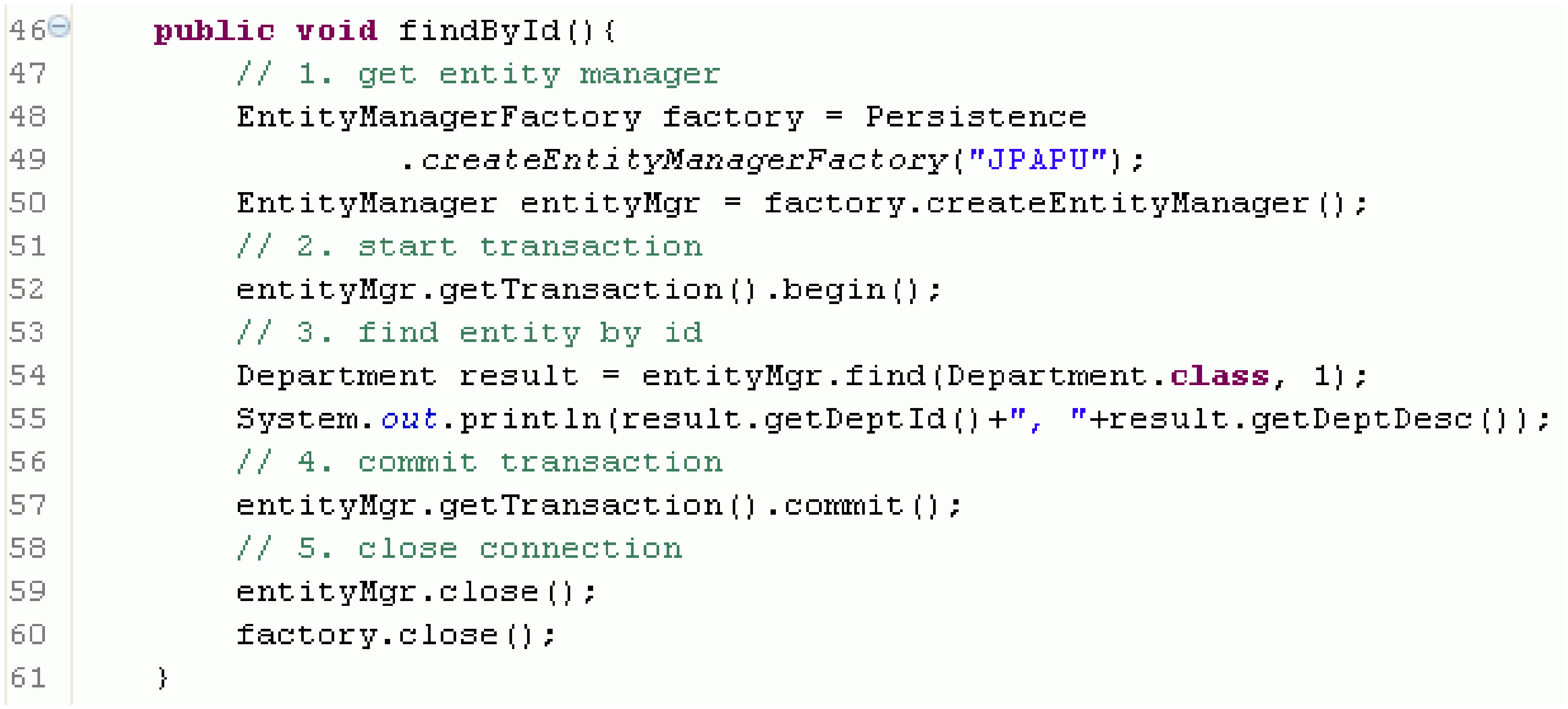


Create Entity



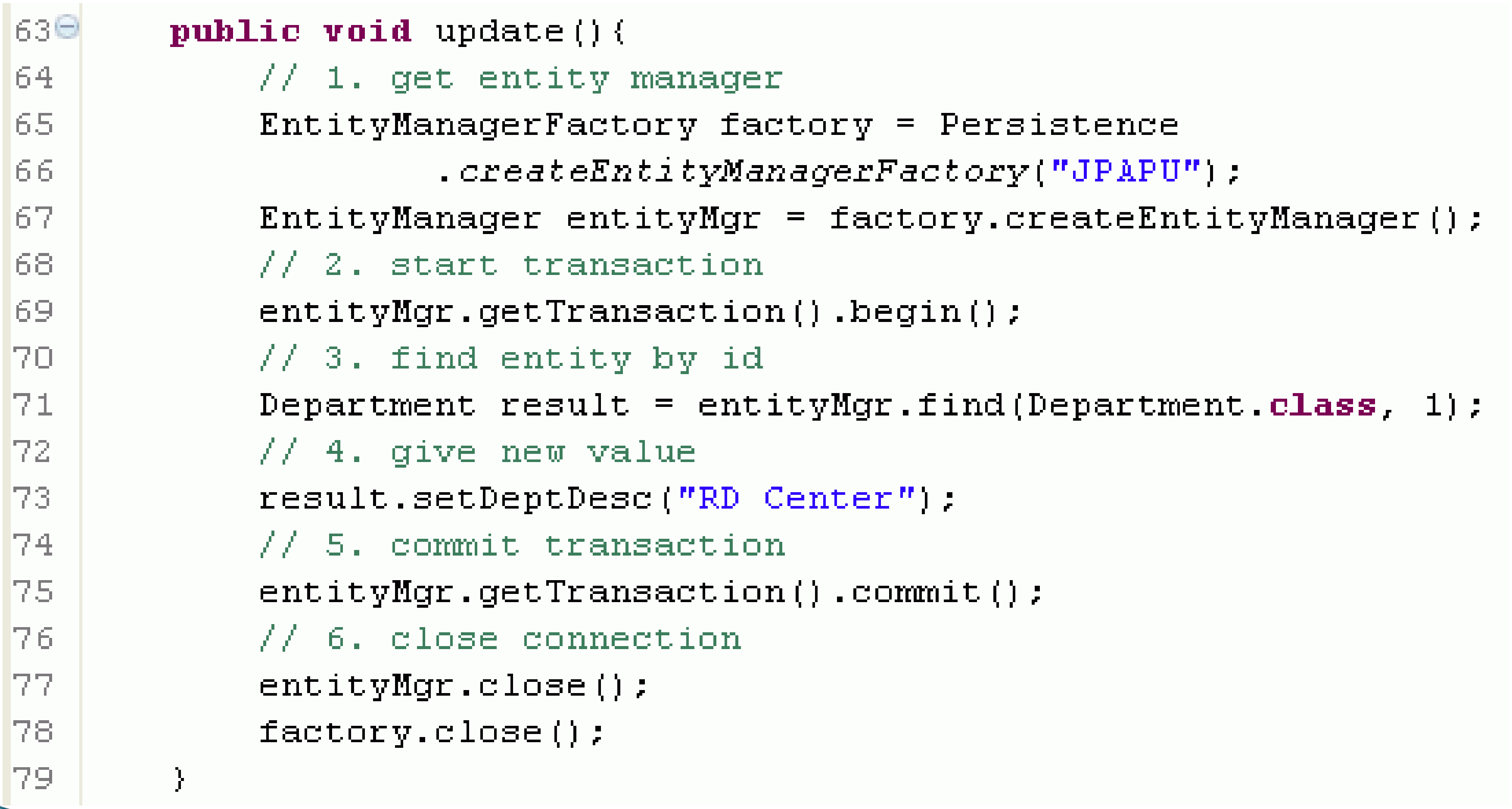
```
public void create(){
    // 1. get entity manager
    EntityManagerFactory factory = Persistence
            .createEntityManagerFactory("JPAPU");
    EntityManager entityMgr = factory.createEntityManager();
    // 2. prepare entity
    Department dept = new Department();
    dept.setDeptId(1);
    dept.setDeptDesc("test");
    // 3. start transaction
    entityMgr.getTransaction().begin();
    // 4. save entity
    entityMgr.persist(dept);
    // 5. commit transaction
    entityMgr.getTransaction().commit();
    // 6. close connection
    entityMgr.close();
    factory.close();
}
```

Find Entity



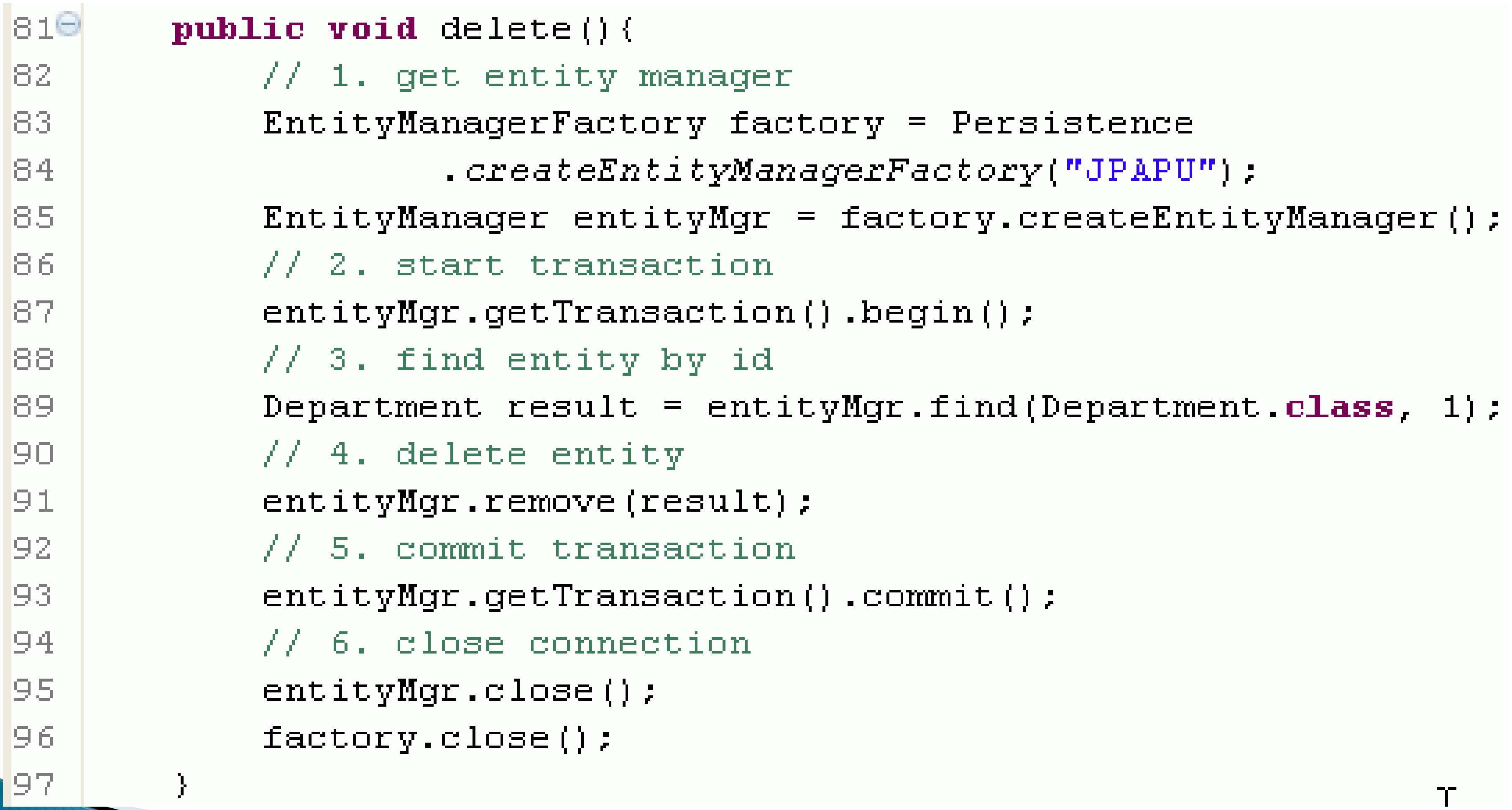
```java
    public void findById(){
        // 1. get entity manager
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory("JPAPU");
        EntityManager entityMgr = factory.createEntityManager();
        // 2. start transaction
        entityMgr.getTransaction().begin();
        // 3. find entity by id
        Department result = entityMgr.find(Department.class, 1);
        System.out.println(result.getDeptId()+", "+result.getDeptDesc());
        // 4. commit transaction
        entityMgr.getTransaction().commit();
        // 5. close connection
        entityMgr.close();
        factory.close();
    }
```

Update Entity



```
public void update(){
    // 1. get entity manager
    EntityManagerFactory factory = Persistence
            .createEntityManagerFactory("JPAPU");
    EntityManager entityMgr = factory.createEntityManager();
    // 2. start transaction
    entityMgr.getTransaction().begin();
    // 3. find entity by id
    Department result = entityMgr.find(Department.class, 1);
    // 4. give new value
    result.setDeptDesc("RD Center");
    // 5. commit transaction
    entityMgr.getTransaction().commit();
    // 6. close connection
    entityMgr.close();
    factory.close();
}
```

Delete Entity



```
public void delete(){
    // 1. get entity manager
    EntityManagerFactory factory = Persistence
            .createEntityManagerFactory("JPAPU");
    EntityManager entityMgr = factory.createEntityManager();
    // 2. start transaction
    entityMgr.getTransaction().begin();
    // 3. find entity by id
    Department result = entityMgr.find(Department.class, 1);
    // 4. delete entity
    entityMgr.remove(result);
    // 5. commit transaction
    entityMgr.getTransaction().commit();
    // 6. close connection
    entityMgr.close();
    factory.close();
}
```