

Addressing Business Concerns



Antonio Goncalves

@agoncal | www.antoniogoncalves.org

Previous Module



Context and Dependency Injection

Interceptors

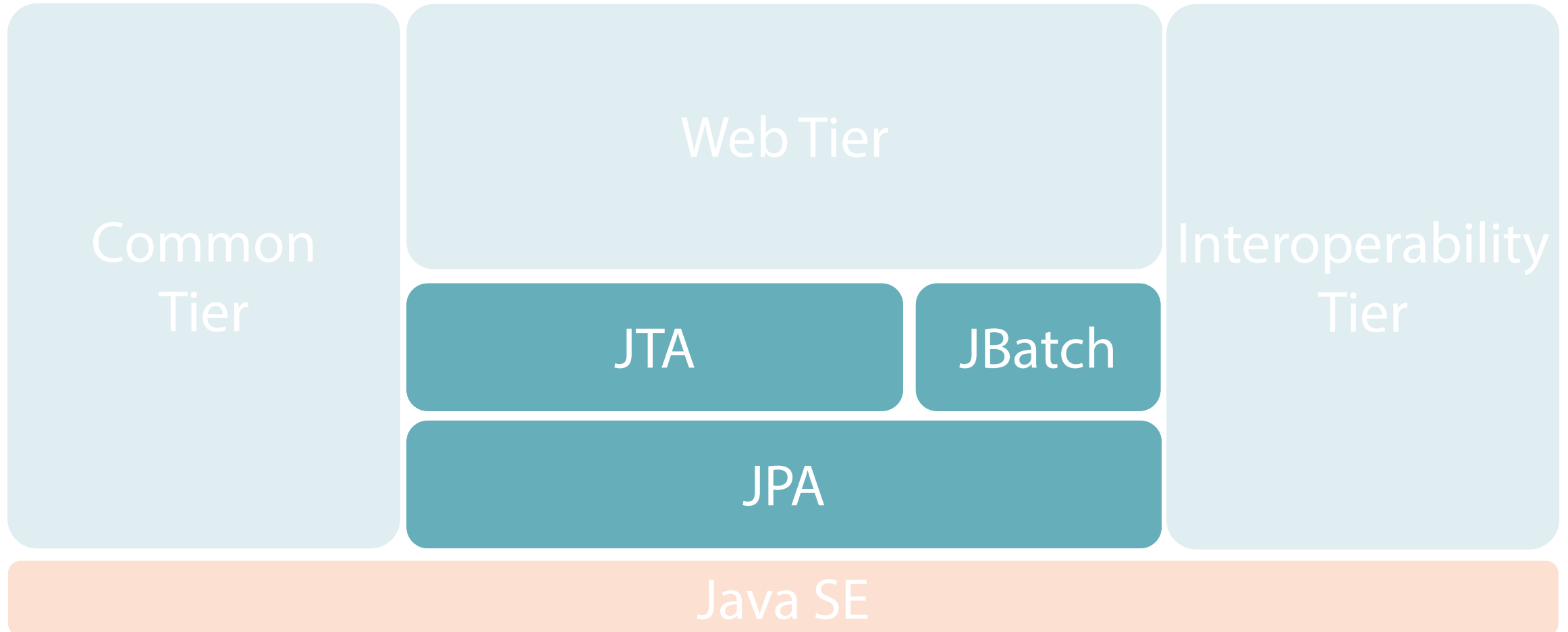
Bean Validation

In most application layers

Module Outline



Module Outline

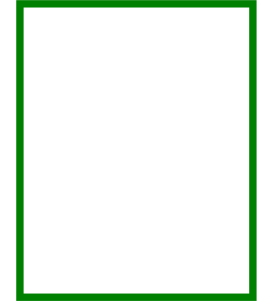


Persistence

Java Persistence API (JPA) 2.1

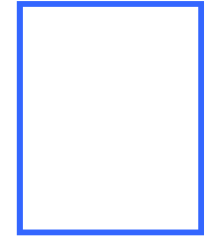
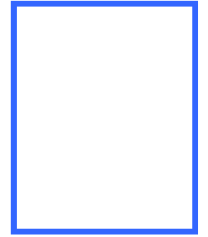
What Is Persistence?

- Data
- Storage
- Central point
- Manipulating data
- Relational databases



When to Use Persistence

- Cookies
- HTTP sessions
- Databases
 - Relational
 - Data warehouses
 - Document-oriented, key-value, graph
- Big data
- Data scientists



JPA Specification

- Java Persistence API 2.1
- JSR 338
- <http://jcp.org/en/jsr/detail?id=338>



Java
Community
Process

The screenshot shows the official page for JSR 338: Java™ Persistence 2.1 on the Java Community Process website. The page is titled "JSRs: Java Specification Requests" and "JSR 338: Java™ Persistence 2.1". It includes a navigation bar with tabs for "JSR", "Community", and "Expert Group". Below the navigation bar, there is a search bar and a list of links for "JSRs by Platform", "JSRs by Technology", "JSRs by Stage", "JSRs by Committee", and "List of All JSRs". The main content area features a table with the following columns: Stage, Access, Start, and Finish. The table lists the stages of the JSR process, from Final Release to JSR Review Ballot, with corresponding access links and dates. Below the table, there is a section for "JCP Info" with links for "About JCP", "Get Involved", "Community Resources", "Community News", "FAQ", and "Contact Us". The page also includes a "Status: Final" section, a "Description" of the Java Persistence API, and an "Expert Group Transparency" section.

Stage	Access	Start	Finish
Final Release	Download page	22 May, 2013	
Final Approval Ballot	View results	09 Apr, 2013	22 Apr, 2013
Proposed Final Draft	Download page	01 Mar, 2013	
Public Review Ballot	View results	22 Jan, 2013	04 Feb, 2013
Public Review	Download page	21 Dec, 2012	21 Jan, 2013
Early Draft Review 2	Download page	31 Oct, 2012	30 Nov, 2012
Early Draft Review	Download page	23 Dec, 2011	22 Jan, 2012
Expert Group Formation		25 Jan, 2011	27 May, 2011
JSR Review Ballot	View results	11 Jan, 2011	24 Jan, 2011

Status: Final
JCP version in use: 2.8
Java Specification Participation Agreement version in use: 2.0

Description:
The Java Persistence API is the Java API for the management of persistence and object/relational mapping in Java EE and Java SE environments.

Expert Group Transparency:
Public Communications
Issue Tracking

JPA Implementations

- EclipseLink
- Hibernate
- Open JPA

The logo for EclipseLink, featuring the word "eclipse" in a purple sans-serif font, followed by a purple crescent moon icon, and the word "link" in a blue sans-serif font.The logo for Hibernate, featuring a hexagonal icon composed of three overlapping triangles in shades of grey and gold, followed by the word "HIBERNATE" in a bold, uppercase, sans-serif font. The letters "HIBERN" are in grey and "ATE" is in gold.The logo for OpenJPA, featuring a stylized "O" made of two overlapping circles in maroon and grey, with a small arrow pointing upwards. To the right of the "O" is the word "penJPA" in a bold, italicized, sans-serif font. The "pen" is in maroon and "JPA" is in black. The entire logo is set against a light grey feather-like background.

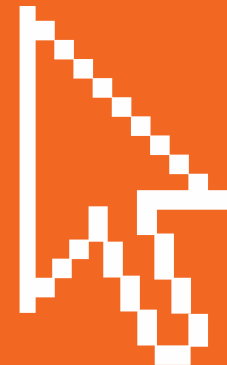
Persistence

Persist data

Search

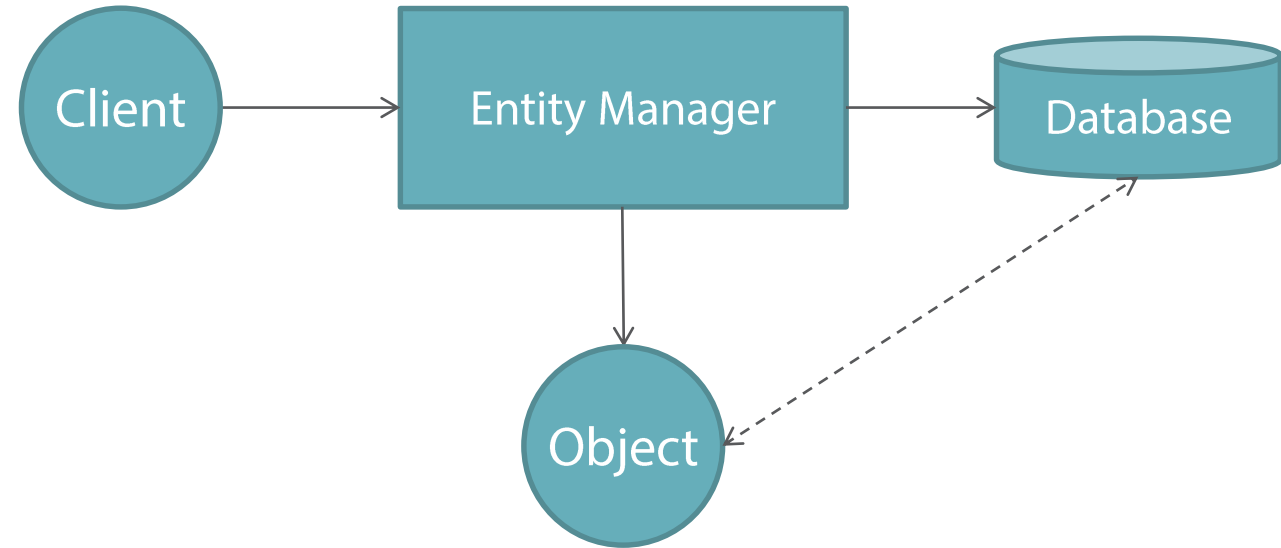
Retrieve

Delete



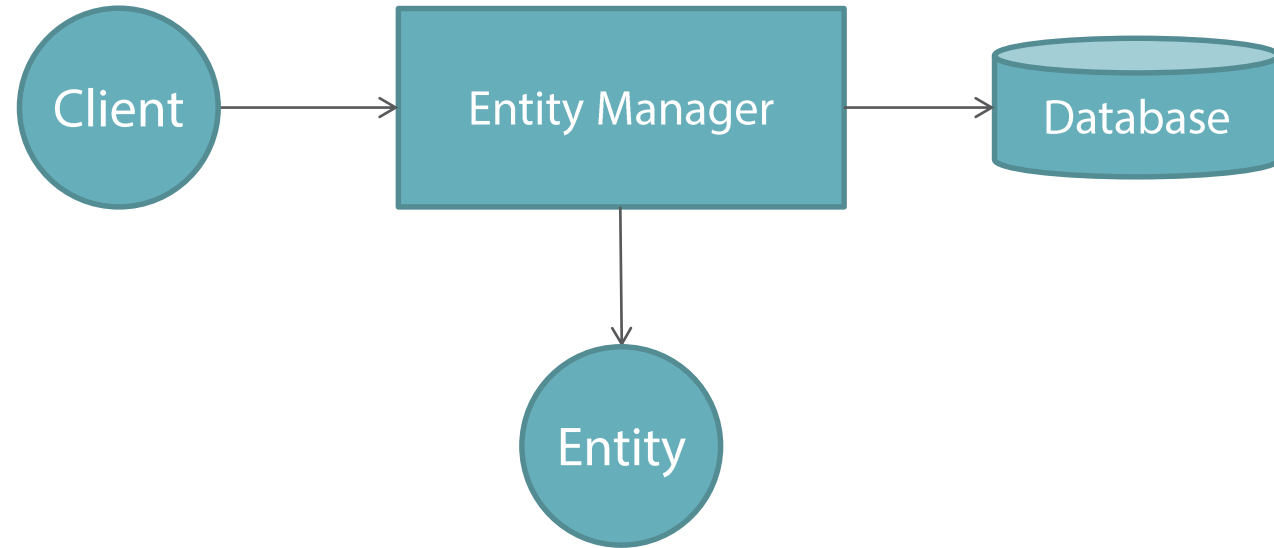
Understanding JPA

- Relational database
- Object
- Object-Relational Mapping (ORM)
- Entity manager



Understanding JPA

- Relational database
- Object
- Object-Relational Mapping (ORM)
- Entity manager
- Entity
- CRUD operations
- Query language JPQL



Entity

- Entity
 - Is an object
 - Live shortly in memory
 - Live persistently in database
 - Mapped to a database
 - Persistent identity
- Object
 - Just live in memory



Entity

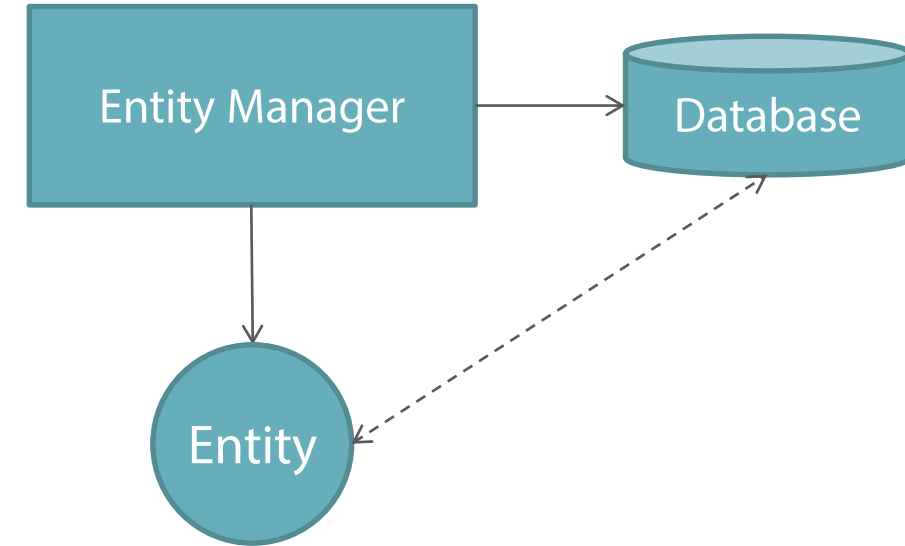
```
@Entity
public class Book {

    @Id
    private Long id;
    private String title;
    private String description;
    private Float unitCost;
    private String isbn;

    // Constructors, getters & setters
}
```

Entity Manager

- Performs database-related operations
- Abstraction above JDBC
- No SQL statements
- CRUD operations through methods
 - (C)reate
 - (R)ead
 - (U)pdate
 - (D)elele



Entity Manager

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book createBook(Book book) {  
        em.persist(book);  
    }  
  
    private Book findBook(Long id) {  
        return em.find(Book.class, id);  
    }  
}
```


Deployment Descriptor

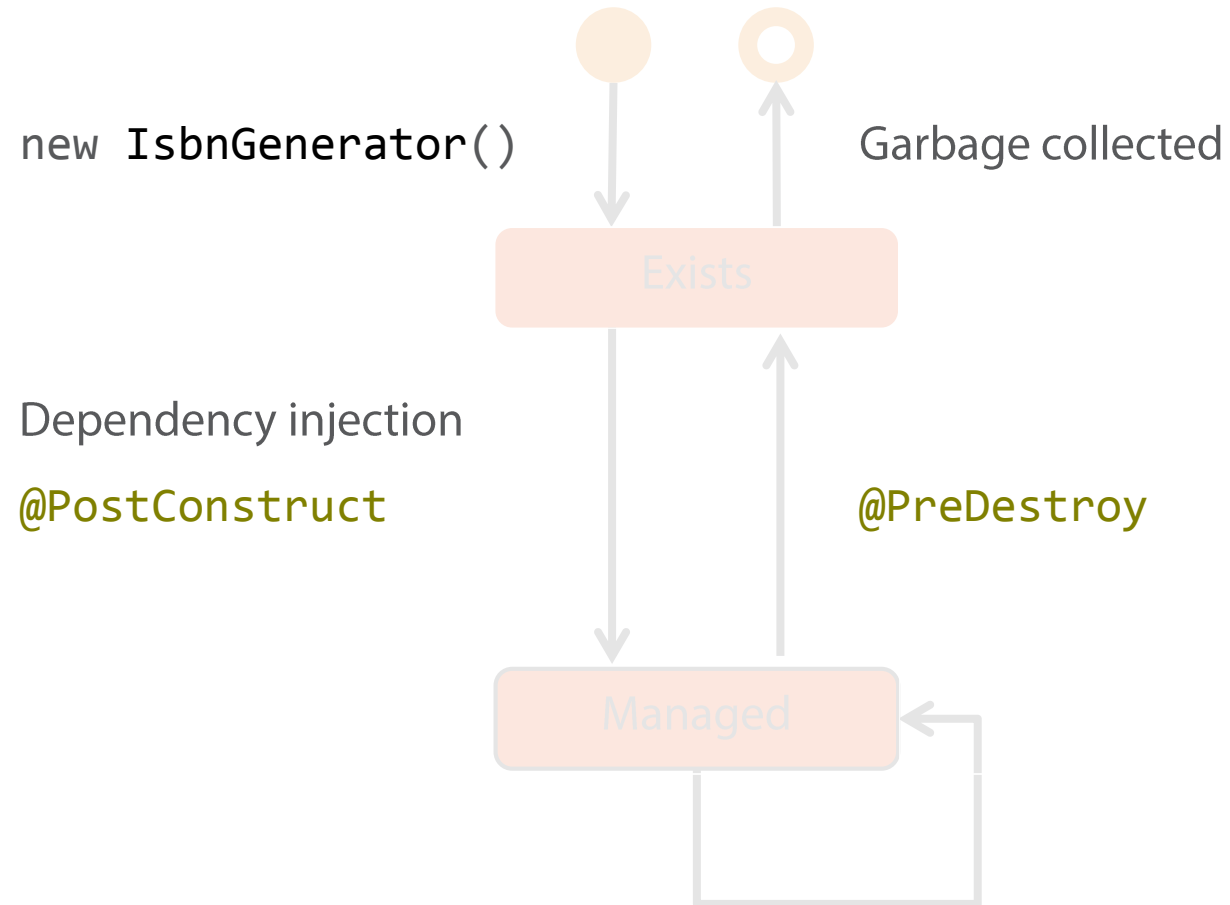
```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

  <persistence-unit name="myPU" transaction-type="JTA">
    <properties>
      <property name="javax.persistence.jdbc.driver"
                value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
                value="jdbc:derby://localhost:1527/myDB"/>
      <property name="javax.persistence.jdbc.user" value="app"/>
      <property name="javax.persistence.jdbc.password" value="app"/>
    </properties>
  </persistence-unit>
</persistence>
```

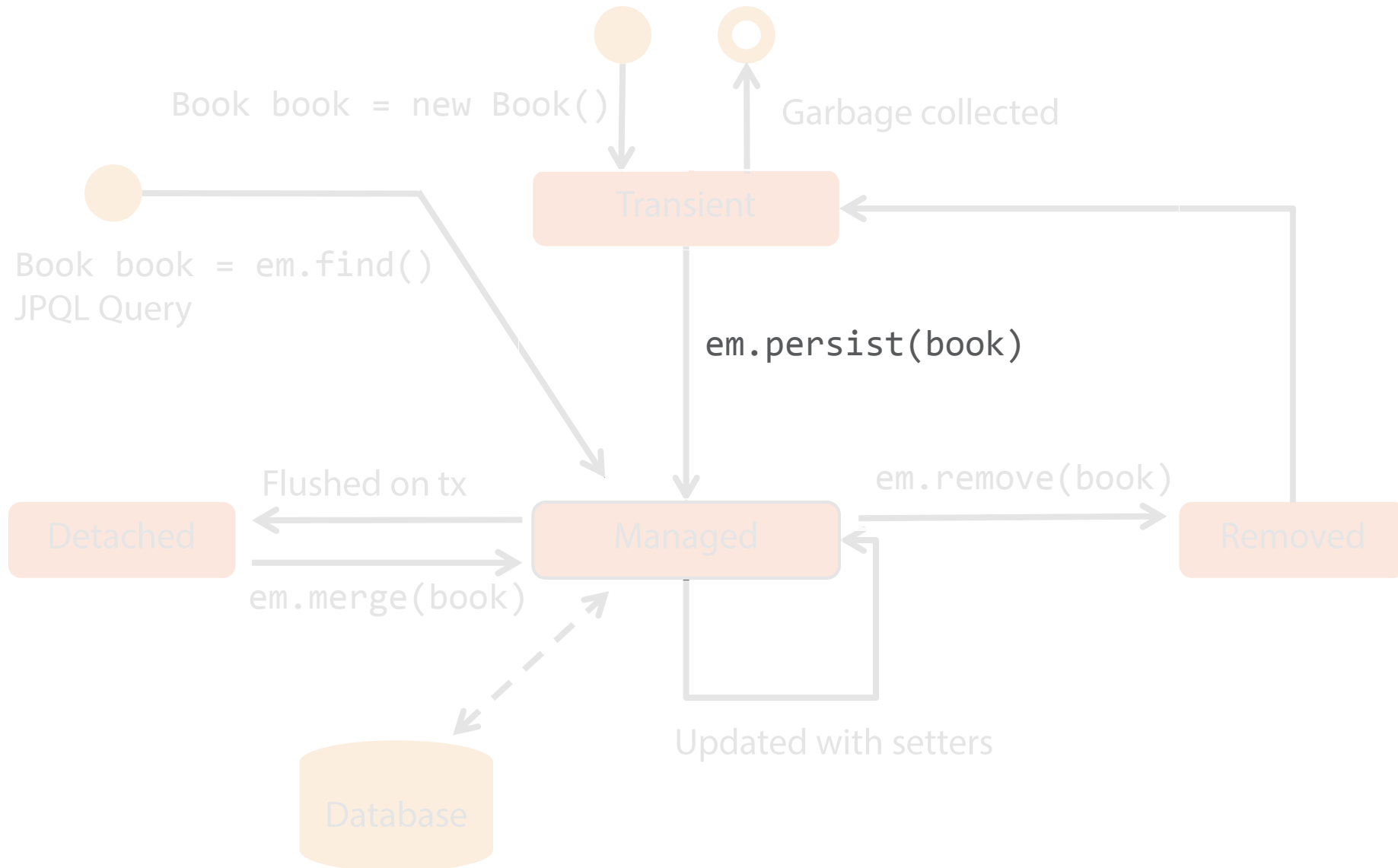
persistence.xml



Life Cycle of Most Java EE Components



Life Cycle of an Entity



Life Cycle of an Entity

@Entity

```
public class Book {
```

@Id

```
private Long id;
```

```
private String title;
```

```
private String description;
```

@PrePersist

@PreUpdate

```
private void validate() {
```

```
    if (title == null || "".equals(title))
```

```
        throw new IllegalArgumentException("Invalid title");
```

```
    if (description == null || "".equals(description))
```

```
        throw new IllegalArgumentException("Invalid description");
```

```
}
```

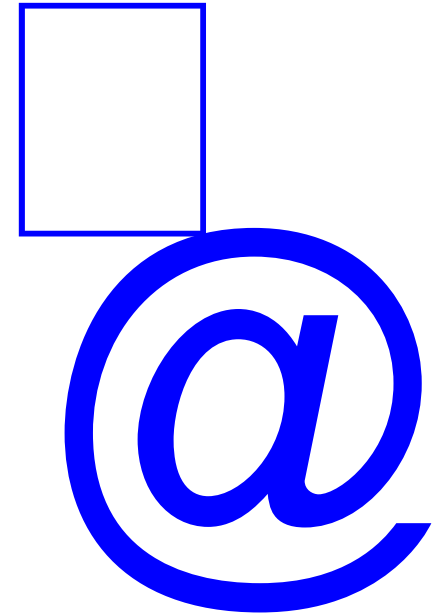
```
}
```

JPA Packages

➔	Package	Description
➔	javax.persistence	Core API
➔	javax.persistence.criteria	Criteria API
➔	javax.persistence.metamodel	Metamodel API
➔	javax.persistence.spi	SPI for JPA providers

Mapping Entities

- Object-relational mapping
- Metadata
 - Annotations
 - XML configuration
- Removes boiler plate code
- Simple mapping
- Complex and flexible mapping



Mapping Entities

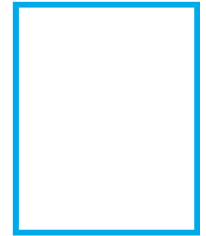
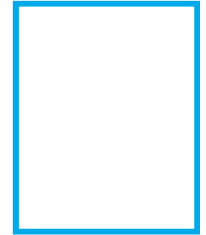
```
@Entity
public class Author {

    @Id
    private Long id;
    private String firstName;
    private String lastName;
    private String bio;
    private Date dateOfBirth;
    private Integer age;
    private Language language;








    // Constructors, getters & setters
}
```

Configuration by Exception

- I.e. convention over configuration
- JPA provider applies the default rules
- Entity name \Leftrightarrow Table name
- Attribute name \Leftrightarrow Column name
- JDBC rules for mapping primitives
 - String \Leftrightarrow VARCHAR(255)
 - Long \Leftrightarrow BIGINT
 - Boolean \Leftrightarrow SMALLINT
 - ...



Database Representation

AUTHOR			
 ID		bigint	
 AGE		integer	N
 BIO		varchar(255)	N
 DATEOFBIRTH		date	N
 FIRSTNAME		varchar(255)	N
 LANGUAGE		integer	N
 LASTNAME		varchar(255)	N

Customize Mapping with Metadata

@Entity

```
public class Author {
```

@Id

```
private Long id;
```

```
private String firstName;
```

```
private String lastName;
```

```
private String bio;
```

```
private Date dateOfBirth;
```

```
private Integer age;
```

```
private Language language;
```

Customizing Table

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Table

```
@Entity
@Table(name = "T_AUTHOR", catalog = "CAT", schema = "APP")
public class Author {

    @Id
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```


Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Generated Value

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;

    private String firstName;

    private String lastName;

    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Columns

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", nullable = true)
    private String lastName;
    @Column(length = 5000)
    private String bio;

    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Temporal Types

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Temporal Types

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.TIME)
    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Temporal Types

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.TIMESTAMP)
    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Temporal Types

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;

    private Integer age;

    private Language language;
```

Customizing Transient State

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;
    @Transient
    private Integer age;

    private Language language;
```


Customizing Enumeration

```
@Entity
@Table(name = "T_AUTHOR")
public class Author {








    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;
    @Transient
    private Integer age;
    @Enumerated(EnumType.ORDINAL)
    private Language language;
```

Customizing Enumeration







```
@Entity
@Table(name = "T_AUTHOR")
public class Author {

    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", length = 50)
    private String firstName;
    @Column(name = "last_name", length = 50)
    private String lastName;
    @Column(length = 5000)
    private String bio;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;
    @Transient
    private Integer age;
    @Enumerated(EnumType.STRING)
    private Language language;
```

Default Database Representation

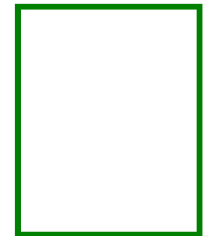
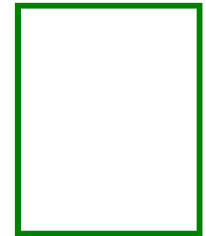
AUTHOR			
	ID	bigint	
	AGE	integer	N
	BIO	varchar(255)	N
	DATEOFBIRTH	date	N
	FIRSTNAME	varchar(255)	N
	LANGUAGE	integer	N
	LASTNAME	varchar(255)	N

Customized Database Representation

T_AUTHOR			
	ID	bigint	
	BIO	varchar(5000)	N
	DATE_OF_BIRTH	date	N
	FIRST_NAME	varchar(50)	N
	LANGUAGE	varchar(255)	N
	LAST_NAME	varchar(255)	

More Mapping

- Relationships
 - Direction
 - Cardinality (one to one, one to many...)
 - Join tables and join columns
- Inheritance
- Mapping using XML instead of annotations



Persistence

Domain model

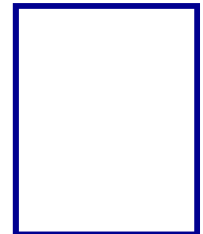
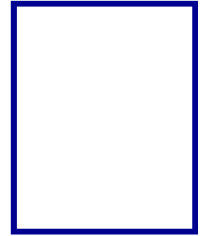
Annotations

Database structure



Querying Entities

- Entity manager
- API for CRUD operations
- Queries
- Manage state and life cycle
- Application obtains the EntityManager
- Using injection



A Book Entity

```
@Entity
public class Book {

    @Id
    private Long id;
    private String title;
    private String description;
    private Float unitCost;

    // Constructors, getters & setters
}
```


Persisting an Entity

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book createBook(Long id, String title, String description, Float unitCost){  
        Book book = new Book();  
        book.setId(id);  
        book.setTitle(title);  
        book.setDescription(description);  
        book.setUnitCost(unitCost);  
  
        em.persist(book);  
  
        return book;  
    }  
}
```

Persisting an Entity

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book createBook(Book book) {  
  
        em.persist(book);  
  
        return book;  
    }  
}
```

Finding by Id

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book findBook(Long id) {  
        return em.find(Book.class, id);  
    }  
  
}
```

Removing an Entity by Id

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public void removeBook(Long id) {  
        Book book = em.find(Book.class, id);  
        if (book != null)  
            em.remove(book);  
    }  
  
}
```

Removing an Entity

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public void removeBook(Book book) {  
        Book bookToBeDeleted = em.merge(book);  
        em.remove(bookToBeDeleted);  
    }  
  
}
```

Removing an Entity

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public void removeBook(Book book) {  
        em.remove(em.merge(book));  
    }  
  
}
```

Updating an Entity

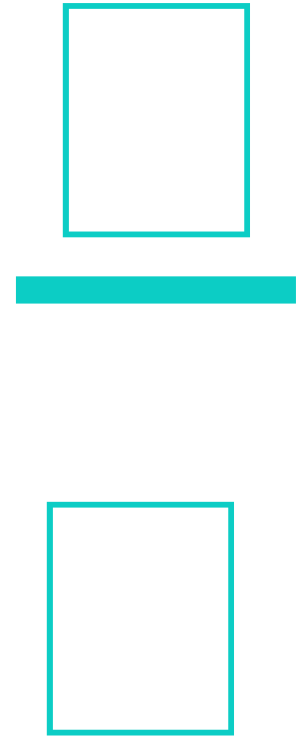
```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book raiseUnitCost(Long id, Float raise) {  
        Book book = em.find(Book.class, id);  
        if (book != null)  
            book.setUnitCost(book.getUnitCost() + raise);  
  
        return book;  
    }  
  
}
```

Updating an Entity

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public Book raiseUnitCost(Book book, Float raise) {  
        Book bookToBeUpdated = em.merge(book);  
        bookToBeUpdated.setUnitCost(bookToBeUpdated.getUnitCost() + raise);  
  
        return book;  
    }  
  
}
```

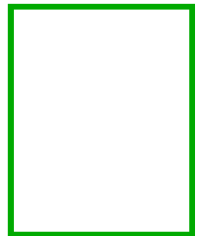
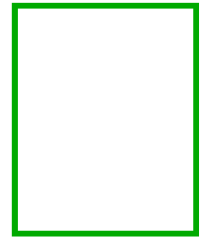

Queries

- Getting data out of the database is crucial
- Search
- Sort
- Aggregate
- Analyze
- Reporting
- Business intelligence



Java Persistence Query Language

- Manipulate entities individually
- Finding by ID is limiting
- Retrieve an entity by criteria
- Retrieve a set of entities
- Query language
- JPQL



JPQL Syntax

SELECT <select clause>
FROM <from clause>
[WHERE <where clause>
[ORDER BY <order by clause>
[GROUP BY <group by clause>
[HAVING <having clause>

<function> **AVG, COUNT, MAX, MIN, SUM**

<operators> **=, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] IN,
[NOT] LIKE, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]**

<num exp.> **ABS, SQRT, MOD, SIZE, INDEX**

<string exp.> **CONCAT, SUBSTRING, TRIM, LOWER, UPPER, LENGTH, LOCATE**

<date exp.> **CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP**

Simplest JPQL Query

```
SELECT    b  
FROM      Book b  
WHERE     b.unitCost > 100
```

Select Clause

```
SELECT  b  
FROM    Book b
```

Select Clause

```
SELECT  b.title, b.unitCost, b.isbn  
FROM    Book b
```

Select Clause

```
SELECT COUNT(b)  
FROM   Book b
```

Select Clause

```
SELECT  AVG(b.unitCost)  
FROM    Book b
```


Select Clause

```
SELECT    b.publisher  
FROM      Book b
```

Select Clause

```
SELECT    b.publisher.name  
FROM      Book b
```

Select Clause

```
SELECT  DISTINCT(b.publisher.name)  
FROM    Book b
```

From Clause

```
SELECT    b  
FROM      Book b
```

Where Clause

```
SELECT    b  
FROM      Book b  
WHERE     b.unitCost > 29
```

Where Clause

```
SELECT    b
FROM      Book b
WHERE     b.unitCost > 29 AND b.nbOfPage < 100
```

Where Clause

```
SELECT    b
FROM      Book b
WHERE     b.unitCost > 29 AND b.nbOfPage BETWEEN 50 AND 90
```

Where Clause

```
SELECT    b  
FROM      Book b  
WHERE     b.title LIKE '%java%'
```


Where Clause

```
SELECT    b  
FROM      Book b  
WHERE     LOWER(b.title) LIKE '%java%'
```

Order By Clause

```
SELECT      b
FROM        Book b
WHERE       LOWER(b.title) LIKE '%java%'
ORDER BY    b.title
```

Order By Clause

```
SELECT    b
FROM      Book b
WHERE     LOWER(b.title) LIKE '%java%'
ORDER BY  b.title ASC
```

Order By Clause

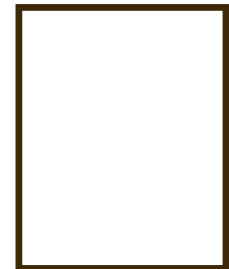
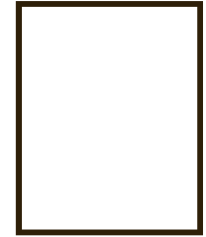
```
SELECT    b
FROM      Book b
WHERE     LOWER(b.title) LIKE '%java%'
ORDER BY  b.title DESC
```

Order By Clause

```
SELECT    b
FROM      Book b
WHERE     LOWER(b.title) LIKE '%java%'
ORDER BY  b.title DESC, b.nbOfPage ASC
```

Executing JPQL Queries

- JPQL statements
- TypedQuery
- Control query execution
- Return entities
- Bind parameters
- Pagination



Executing JPQL Queries

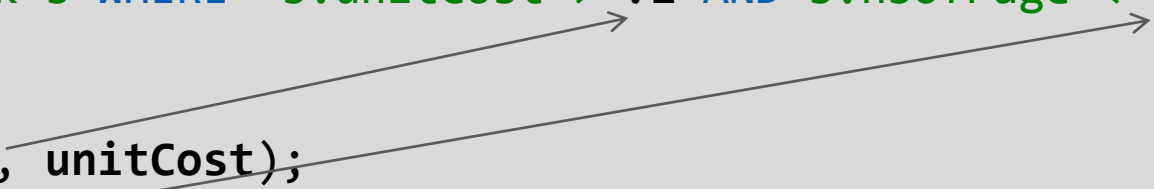
```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public List<Book> findBigBooks() {  
        TypedQuery<Book> query = em.createQuery(  
            "SELECT b FROM Book b WHERE b.unitCost > 29 AND b.nbOfPage < 700",  
            Book.class);  
  
        List<Book> books = query.getResultList();  
  
        return books;  
    }  
}
```

Binding Parameters

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public List<Book> findBigBooks() {  
        TypedQuery<Book> query = em.createQuery(  
            "SELECT b FROM Book b WHERE b.unitCost > 29 AND b.nbOfPage < 700",  
            Book.class);  
  
        List<Book> books = query.getResultList();  
  
        return books;  
    }  
}
```


Binding Parameters


```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public List<Book> findBigBooks(Float unitCost, Float nbOfPage) {  
        TypedQuery<Book> query = em.createQuery(  
            "SELECT b FROM Book b WHERE b.unitCost > ?1 AND b.nbOfPage < ?2",  
            Book.class);  
  
        query.setParameter(1, unitCost);  
        query.setParameter(2, nbOfPage);  
  
        List<Book> books = query.getResultList();  
  
        return books;  
    }  
}
```



The diagram illustrates the binding of parameters from the method arguments to the SQL query placeholders. Two arrows originate from the method arguments: one from `unitCost` pointing to the `?1` placeholder in the SQL query, and another from `nbOfPage` pointing to the `?2` placeholder. This demonstrates how the values passed to the method are mapped to the query parameters.

Binding Parameters

```
public class BookService {  
  
    @PersistenceContext(unitName = "myPU")  
    private EntityManager em;  
  
    public List<Book> findBigBooks(Float unitCost, Float nbOfPage) {  
        TypedQuery<Book> query = em.createQuery(  
            "SELECT b FROM Book b WHERE b.unitCost > :cost AND b.nbOfPage < :pages",  
            Book.class);  
  
        query.setParameter("cost", unitCost);  
        query.setParameter("pages", nbOfPage);  
  
        List<Book> books = query.getResultList();  
  
        return books;  
    }  
}
```



The diagram illustrates the binding of parameters from the Java code to the SQL query. Two arrows originate from the `setParameter` calls in the Java code and point to the corresponding placeholders in the SQL query string. The first arrow points from `query.setParameter("cost", unitCost);` to the `:cost` placeholder in the SQL query. The second arrow points from `query.setParameter("pages", nbOfPage);` to the `:pages` placeholder in the SQL query.

Persistence

Entity manager

CRUD operations

JPQL queries

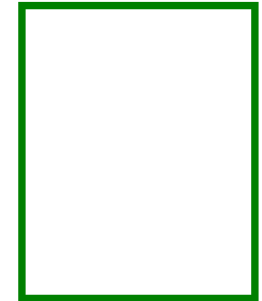


Transactions

Java Transaction API (JTA) 1.2

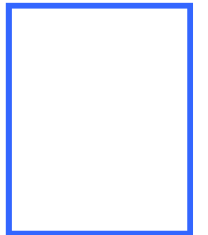
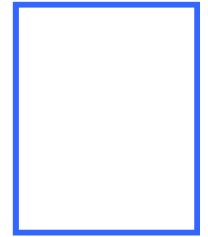
What Are Transactions?

- Data is crucial
- Kept in a consistent state
- Group of operations
- Performed in a single unit of work
 - Persisting data in a database
 - Sending messages
 - Invoking web services...



When to Use Transactions

- Several operations
- All succeed or all fail
- Commit or rollback
- ACID properties
 - Atomicity
 - Consistency
 - Isolation
 - Durability



JTA Specification

- Java Transaction API 1.2
- JSR 907
- <http://jcp.org/en/jsr/detail?id=907>



Java
Community
Process

A screenshot of the Java Community Process website showing the details for JSR 907: Java™ Transaction API (JTA). The page includes a navigation bar with tabs for JSR, Community, and Expert Group. The main content area displays the title "JSRs: Java Specification Requests" and "JSR 907: Java™ Transaction API (JTA)". Below this is a table with columns for Stage, Access, Start, and Finish, listing various milestones from 2000 to 2013. The status is "Maintenance" and the JCP version in use is "2.6". The page also includes a "Team" section with "Specification Leads" and a "Description" section.

The Java Community Process

Community Development of Java Technology Specifications

JSR Community Expert Group

Summary Proposal Detail (Summary & Proposal)

Search JSRs

JSRs: Java Specification Requests
JSR 907: Java™ Transaction API (JTA)

Stage	Access	Start	Finish
Maintenance Release	Download page	17 Jun, 2013	
Maintenance Draft Review 5	Download page	28 Feb, 2013	15 Apr, 2013
Maintenance Draft Review 4	Download page	03 Nov, 2005	05 Dec, 2005
Maintenance Draft Review 3	Download page	04 Aug, 2005	06 Sep, 2005
Final Release	Download page	06 Nov, 2002	
Maintenance Draft Review 2	Download page	01 Oct, 2001	05 Nov, 2001
Maintenance Draft Review	Download page	19 Jul, 2000	18 Aug, 2000

Status: Maintenance
JCP version in use: 2.6
Java Specification Participation Agreement version in use: 2.0

Description:
Revisions to the JTA specification.

Please direct comments on this JSR to the Spec Lead(s)

Team

Specification Leads

Paul Parkinson Oracle

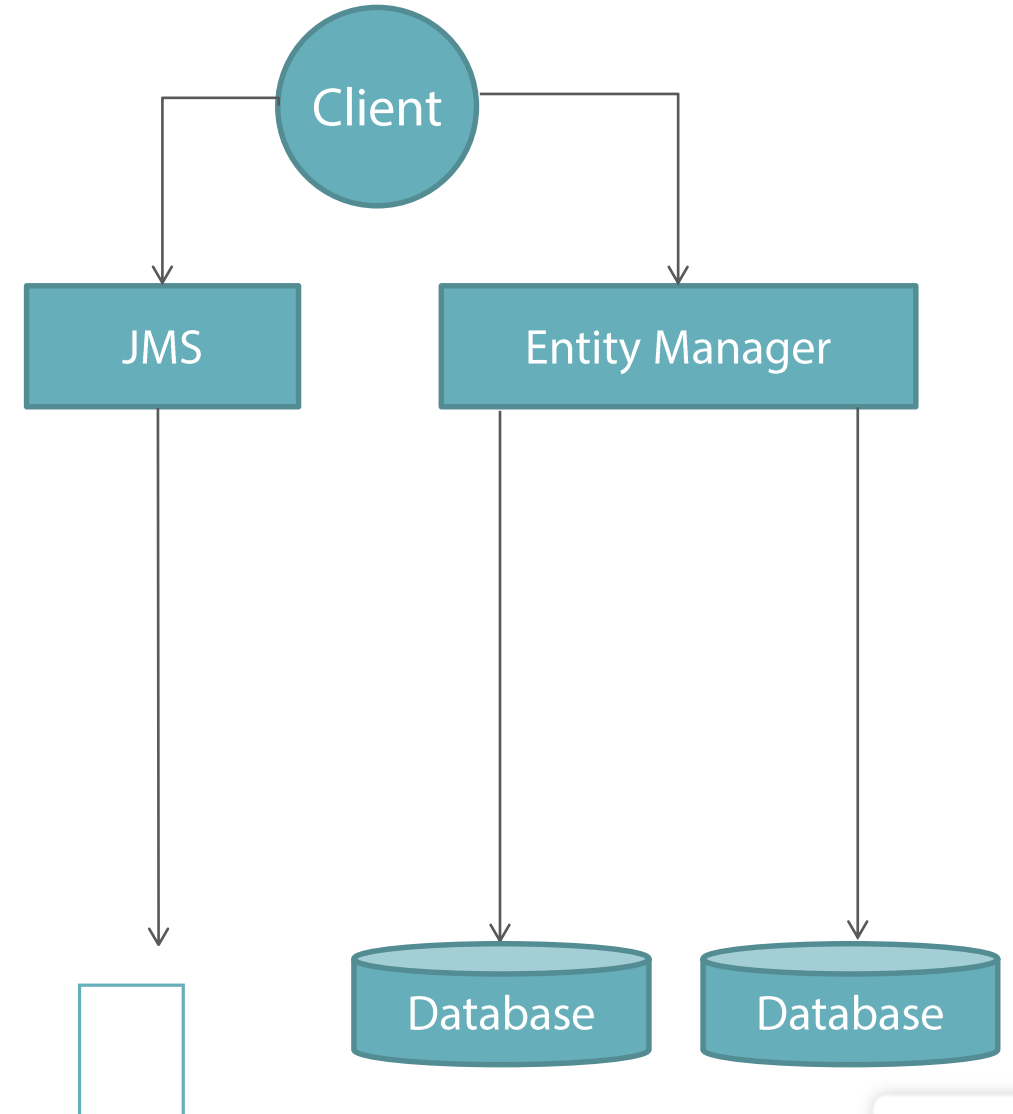
JTA Implementations

- GlassFish Transaction Manager
- JBoss Transaction Manager
- Atomikos
- Bitronix JTA



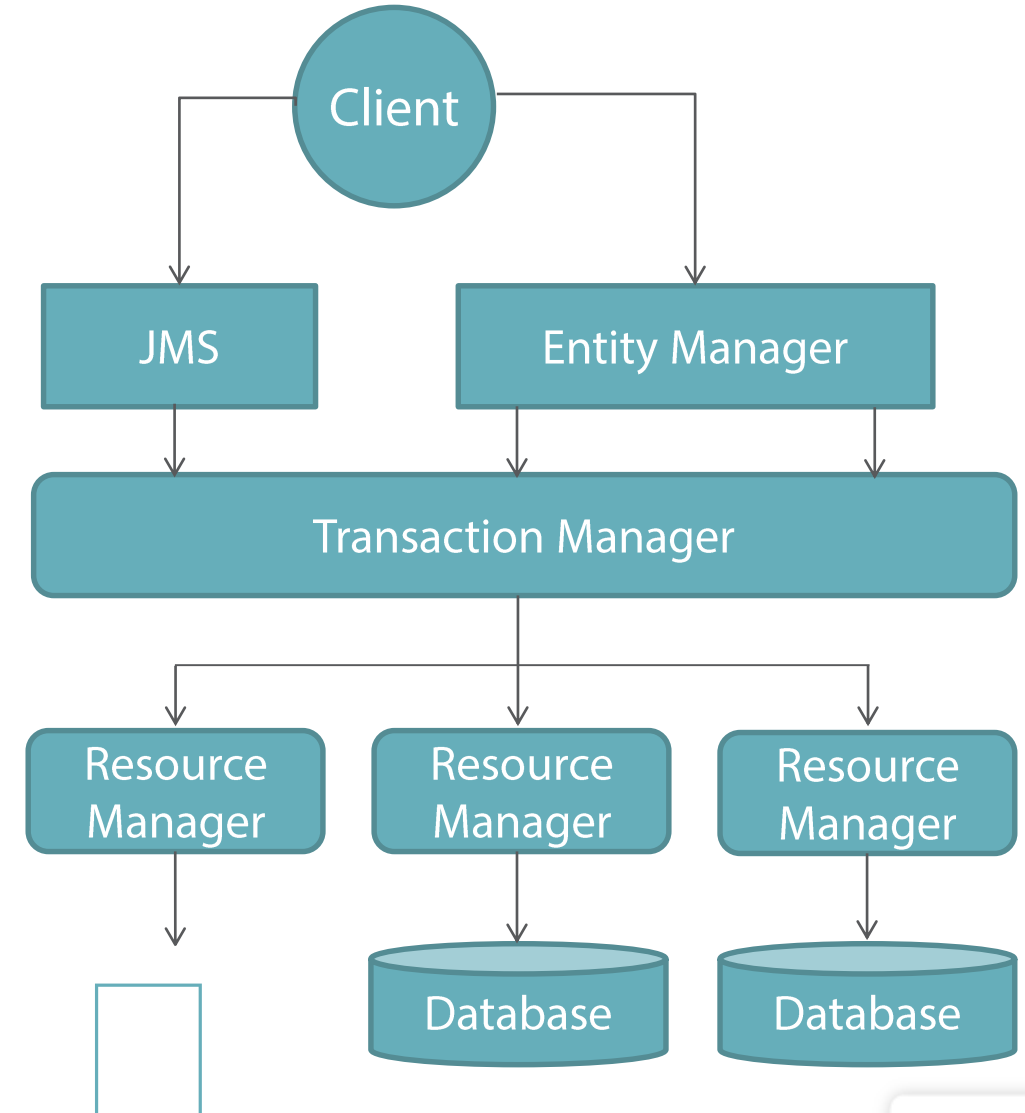
Understanding JTA

- Several databases
- Message Oriented Middleware



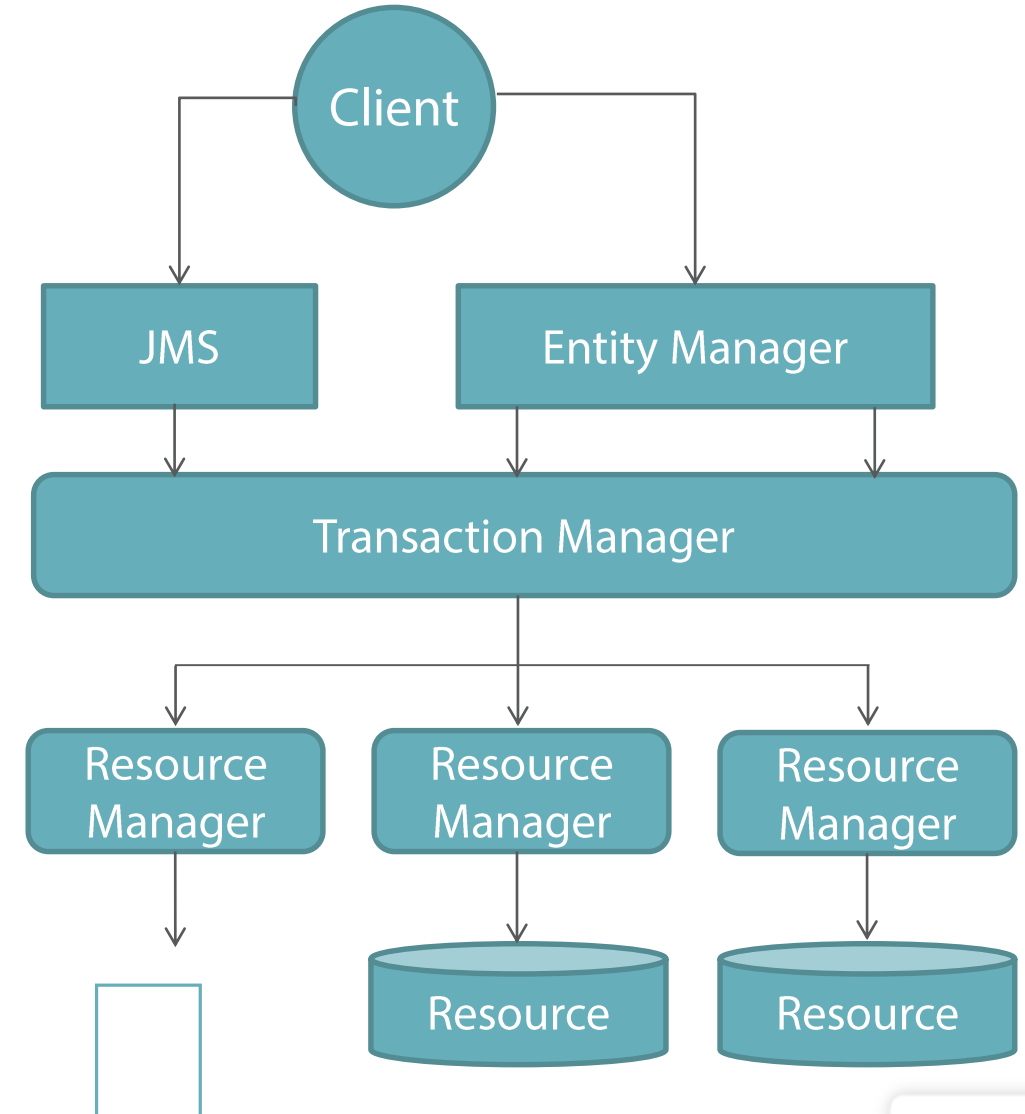
Understanding JTA

- Several databases
- Message Oriented Middleware
- Transaction manager
- Resource manager
- Resource



Understanding JTA

- Several databases
- Message Oriented Middleware
- Transaction manager
- Resource manager
- Resource



Transaction Demarcation



```
@Transactional
public class BookService {

    @PersistenceContext(unitName = "myPU")
    private EntityManager em;

    @Transactional
    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }

    @Transactional
    public void removeBook(Book book) {
        em.remove(em.merge(book));
    }
}
```

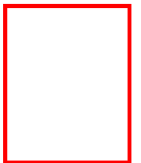
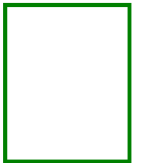
JTA Packages



Package	Description
<code>javax.transaction</code>	Core JTA APIs
<code>javax.transaction.xa</code>	APIs for distributed transactions

Managing Transactions

- Transaction management is low-level
- Better use high-level APIs
- `@Transactional`
- Several operations
- All succeed or all fail
- Rollback on exceptions
- Transactional policy
- Resource neutral



Transaction Demarcation

```
public class BookService {
```

```
    @PersistenceContext
```

```
    private EntityManager em;
```

```
    @Transactional
```

```
    public Book createBook(Book book) {
```

```
        em.persist(book);
```

```
        // Business logic
```

```
        return book;
```

```
    }
```

```
}
```

Starts a transaction



Commit or rollback the transaction



Exceptions and Transactions

```
public class BookService {  
  
    @PersistenceContext  
    private EntityManager em;  
  
    @Transactional  
  
    public Book createBook(Book book) {  
        em.persist(book);  
        // Business logic  
        return book;  
    }  
}
```


Exceptions and Transactions

```
public class BookService {  
  
    @PersistenceContext  
    private EntityManager em;  
  
    @Transactional(dontRollbackOn = { Exception.class })  
  
    public Book createBook(Book book) {  
        em.persist(book);  
        // Business logic  
        return book;  
    }  
}
```

Exceptions and Transactions

```
public class BookService {  
  
    @PersistenceContext  
    private EntityManager em;  
  
    @Transactional(rollbackOn      = { ArrayIndexOutOfBoundsException.class },  
                   dontRollbackOn = { SQLWarning.class, SQLException.class })  
    public Book createBook(Book book) {  
        em.persist(book);  
        // Business logic  
        return book;  
    }  
}
```

Transaction Propagation

```
@Transactional
public class BookService {

    @PersistenceContext
    private EntityManager em;

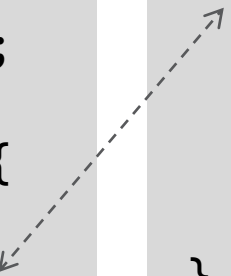
    @Inject
    private InventoryService inventory;

    public Book createBook(Book book) {
        em.persist(book);
        inventory.addItem(book.getId());
        return book;
    }
}
```

```
@Transactional
public class InventoryService {

    @PersistenceContext
    private EntityManager em;

    public void addItem(Long id) {
        Book book = em.find(Book.class, id);
        if (book != null)
            book.updateStock();
    }
}
```



Transactional Policies

	Package	Description
➡	REQUIRED	Always propagates the transaction (default)
➡	REQUIRES_NEW	Creates a new transaction before executing a method
➡	SUPPORTS	Inherits the client's transaction context
➡	MANDATORY	Requires a transaction before invoking the business method
➡	NOT_SUPPORTED	Cannot be invoked in a transaction context
➡	NEVER	Must not be invoked from a transactional client

Required

```
@Transactional
public class BookService {

    @PersistenceContext
    private EntityManager em;

    @Inject
    private InventoryService inventory;

    public Book createBook(Book book) {
        em.persist(book);
        inventory.addItem(book.getId());
        return book;
    }
}
```

```
@Transactional
public class InventoryService {

    @PersistenceContext
    private EntityManager em;

    public void addItem(Long id) {
        Book book = em.find(Book.class, id);
        if (book != null)
            book.updateStock();
    }
}
```

tx1


Requires New

```
@Transactional
public class BookService {

    @PersistenceContext
    private EntityManager em;

    @Inject
    private InventoryService inventory;


    public Book createBook(Book book) {
        em.persist(book);
        inventory.addItem(book.getId());
        return book;
    }
}
```



```
@Transactional(REQUIRES_NEW)
public class InventoryService {

    @PersistenceContext
    private EntityManager em;

    public void addItem(Long id) {
        Book book = em.find(Book.class, id);
        if (book != null)
            book.updateStock();
    }
}
```



Transactions

@Transactional

Exception

Commit or rollback

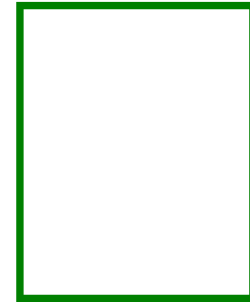


Batch Processing

JBatch 1.0

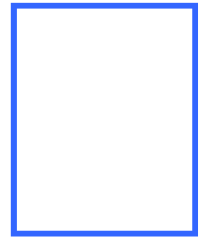
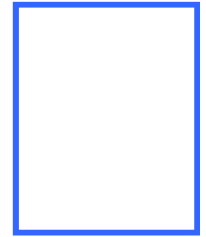
What Is Batch Processing?

- Execution of a series of jobs
- Set of inputs
- Without manual intervention
- High-volume
- Repetitive tasks



When to Use Batch Processing

- Processing high volumes of data
- Over a period of time
 - Payroll
 - Statement generation
 - Interest calculation
 - ETL (Extract, Load, and Transform)
- Run on schedule or on-demand
- Grid computing



JBatch Specification

- JBatch 1.0
- JSR 352
- <http://jcp.org/en/jsr/detail?id=352>



Java
Community
Process

The screenshot shows the JSR 352 page on the Java Community Process website. The page is titled "JSR 352: Batch Applications for the Java Platform" and is part of the "JSRs: Java Specification Requests" section. It includes a table of stages and a description of the specification.

JSRs: Java Specification Requests
JSR 352: Batch Applications for the Java Platform

Stage	Access	Start	Finish
Maintenance Release	Download page	19 Aug, 2014	
Maintenance Review Ballot	View results	17 Jun, 2014	23 Jun, 2014
Maintenance Draft Review	Download page	14 Apr, 2014	13 Jun, 2014
Final Release	Download page	24 May, 2013	
Final Approval Ballot	View results	26 Mar, 2013	08 Apr, 2013
Proposed Final Draft	Download page	17 Jan, 2013	
Public Review Ballot	View results	04 Dec, 2012	17 Dec, 2012
Public Review	Download page	02 Nov, 2012	03 Dec, 2012
Early Draft Review	Download page	30 Aug, 2012	29 Sep, 2012
Expert Group Formation		29 Nov, 2011	08 Feb, 2012
JSR Review Ballot	View results	15 Nov, 2011	28 Nov, 2011
JSR Review		26 Oct, 2011	14 Nov, 2011

Status: Maintenance
JCP version in use: 2.9
Java Specification Participation Agreement version in use: 2.0

Description:
This JSR specifies a programming model for batch applications and a runtime for scheduling and executing jobs.

JBatch Implementations

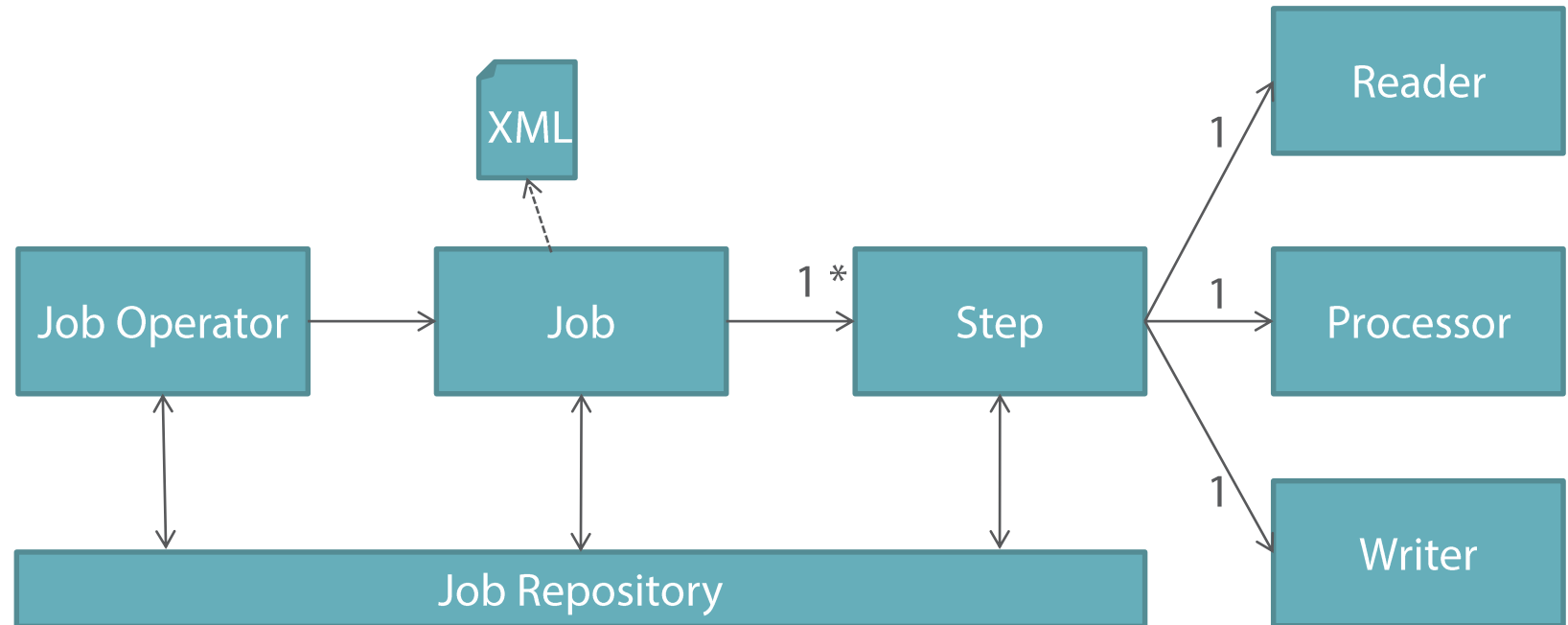
- WebSphere Liberty Batch
- Spring Batch
- JBeret



JBeret

Understanding JBatch

- Job
 - Reader
 - Processor
 - Writer
- Job operator
- Job repository



Job

```
<job id="invoiceJob" (...) version="1.0">
  <step id="invoice">
    <chunk item-count="1">
      <reader ref="invoiceReader">
        <processor ref="invoiceProcessor"/>
        <writer ref="invoiceWriter"/>
      </chunk>
    </step>

    <step id="dispatch"> (...) </step>
  </job>
```

Reader

```
@Named
public class InvoiceReader extends AbstractItemReader {

    @PersistenceContext
    private EntityManager em;

    @Override
    public Object readItem() throws Exception {

        TypedQuery<Invoice> query = em.createQuery(
            "SELECT i FROM Invoice i ORDER BY i.invoiceDate ASC", Invoice.class);
        List<Invoice> invoices = query.getResultList();
        return invoices;
    }
}
```

Processor

@Named

```
public class InvoiceProcessor implements ItemProcessor {
```

@Override

```
public Object processItem(Object item) throws Exception {
```

```
    List<Invoice> invoices = (List<Invoice>) item;
```

```
    List<InvoiceSummary> summaries = new ArrayList<>();
```

```
    // Business logic
```

```
    return summaries;
```

```
}
```

```
}
```


Writer

@Named

```
public class InvoiceWriter extends AbstractItemWriter {
```

@Override

```
public void writeItems(List<Object> items) throws Exception {
```

```
    InvoiceSummaries summaries = items.get(0);
```

```
    summaries.setYear(2016);
```

```
    // Business logic
```

```
    JAXBContext context = JAXBContext.newInstance(InvoiceSummaries.class);
```

```
    Marshaller m = context.createMarshaller();
```

```
    File file = new File("invoice.xml");
```

```
    m.marshal(summaries, file);
```

```
}
```

```
}
```

Starting a Job

```
@WebServlet(name = "InvoiceJobServlet", urlPatterns = "startJob")
public class InvoiceJobServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        JobOperator jobOperator = BatchRuntime.getJobOperator();
        jobOperator.start("invoiceJob", null);
    }
}
```

JBatch Packages

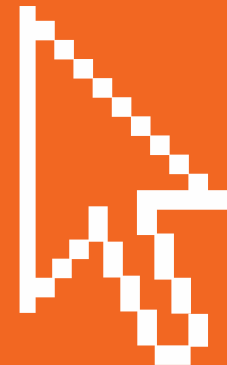
➔	Package	Description
➔	javax.batch.api	Core Batch APIs
➔	javax.batch.operations	Batch operations
	javax.batch.runtime	Batch runtime

Batch Processing

Read invoices from DB

Process invoices

Write XML file



Summary



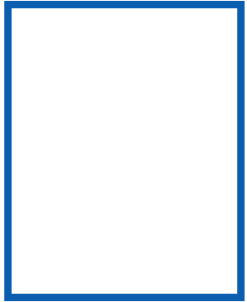
Business layer specifications

Java Persistence API

Java Transaction API

Batch processing

What's Next



Web tier

Servlets

Web pages

Web sockets