Addressing Business Concerns

Antonio Goncalves

@agoncal | www.antoniogoncalves.org

Previous Module

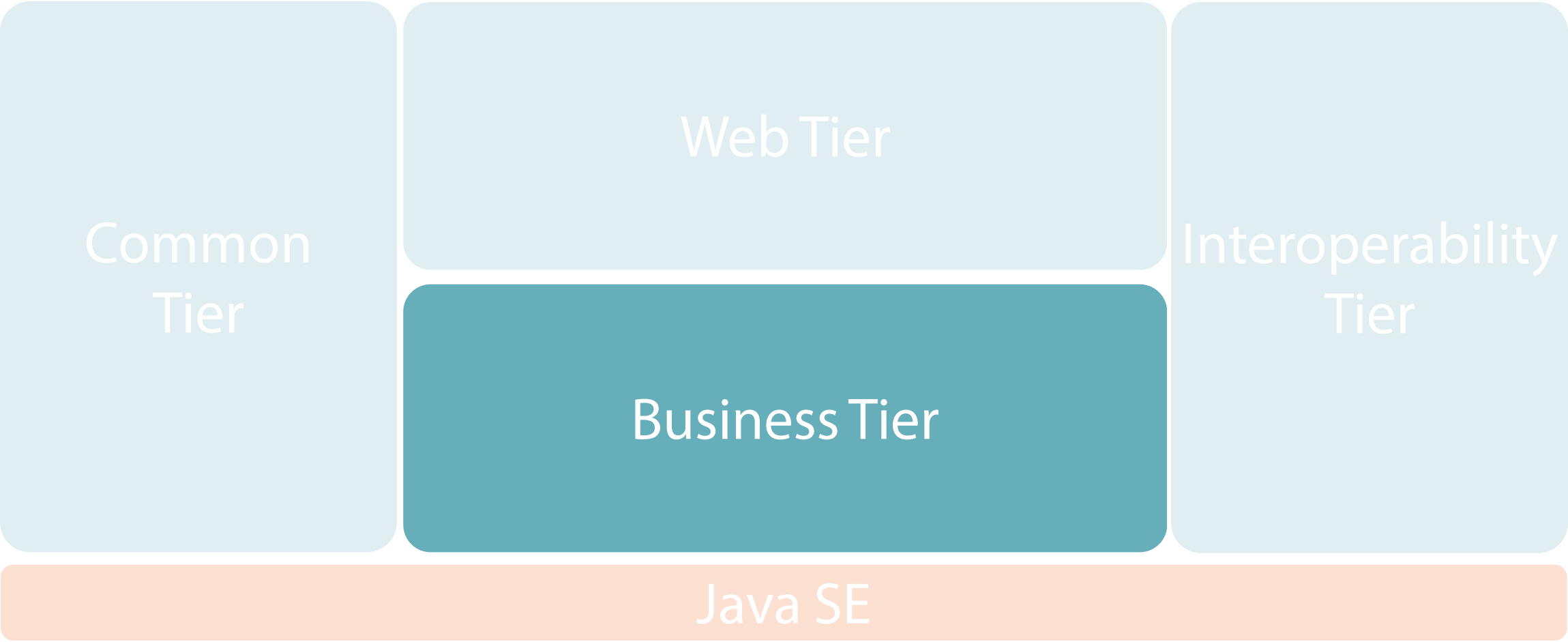
Context and Dependency Injection

Interceptors

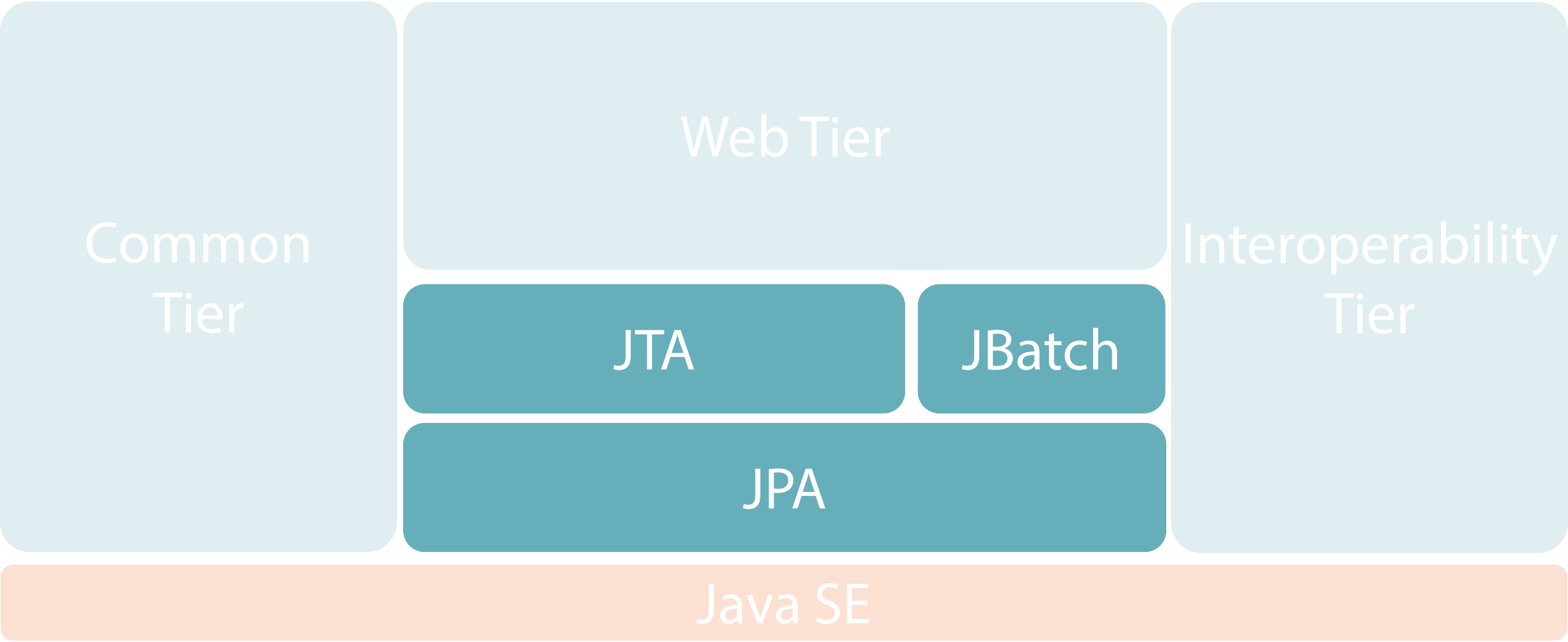
Bean Validation

In most application layers

Module Outline



Module Outline



Persistence

Java Persistence API (JPA) 2.1

What Is Persistence?

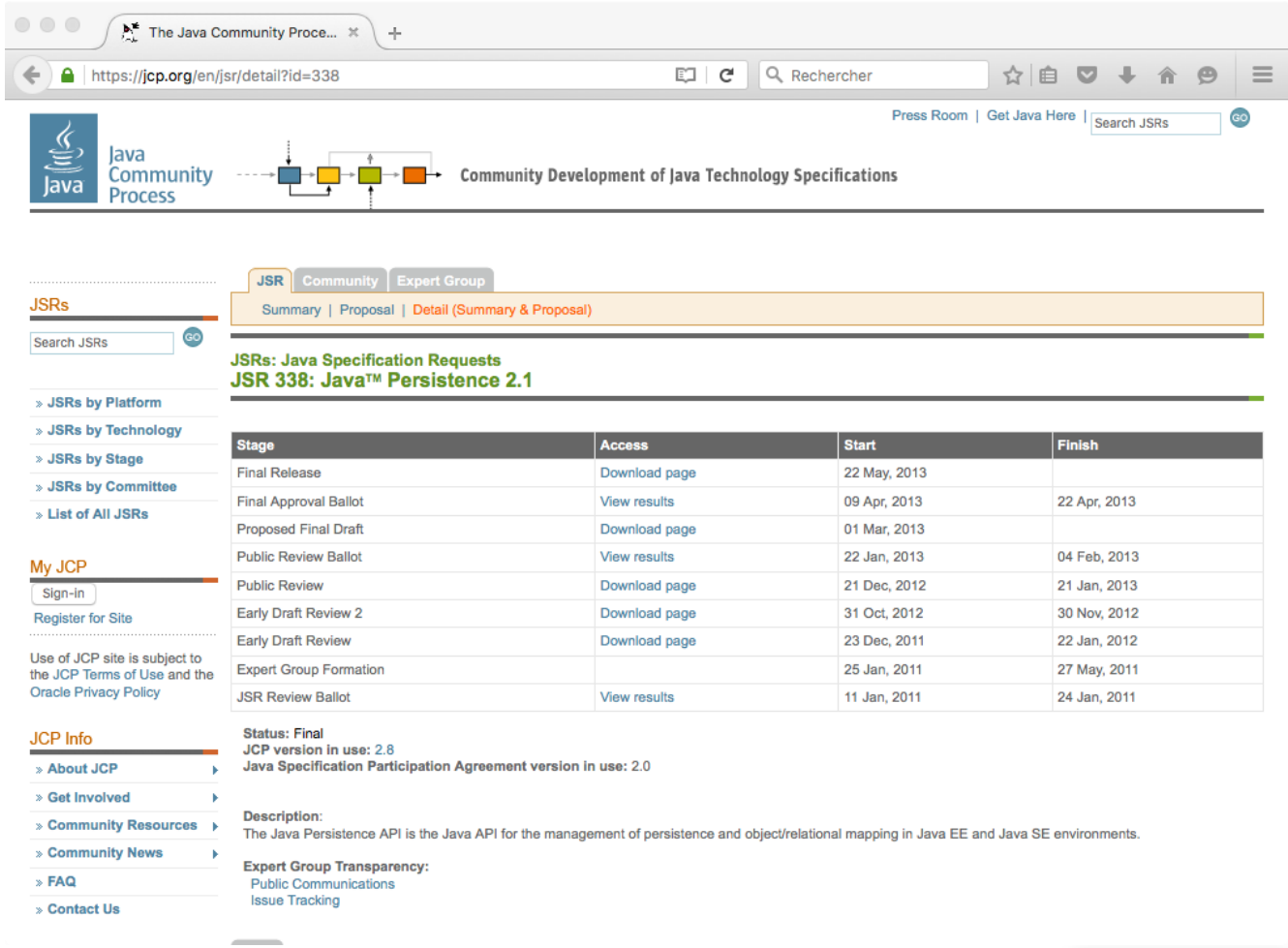
- Data
- Storage
- Central point
- Manipulating data
- Relational databases

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

JPA Implementations

- EclipseLink
- Hibernate
- Open JPA

Persistence

Persist data

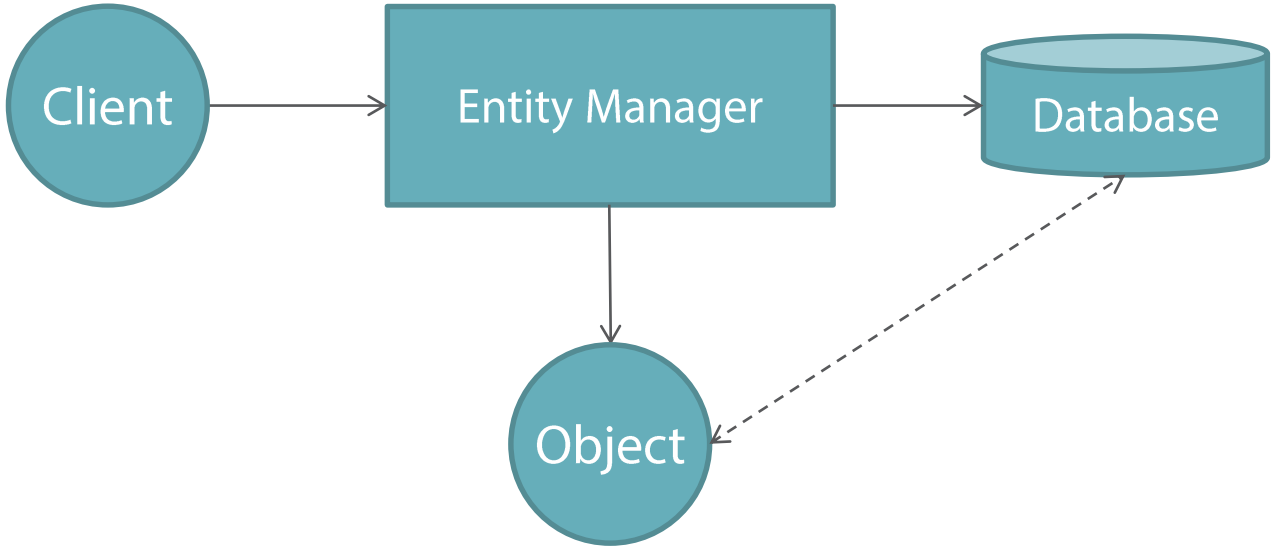
Search

Retrieve

Delete

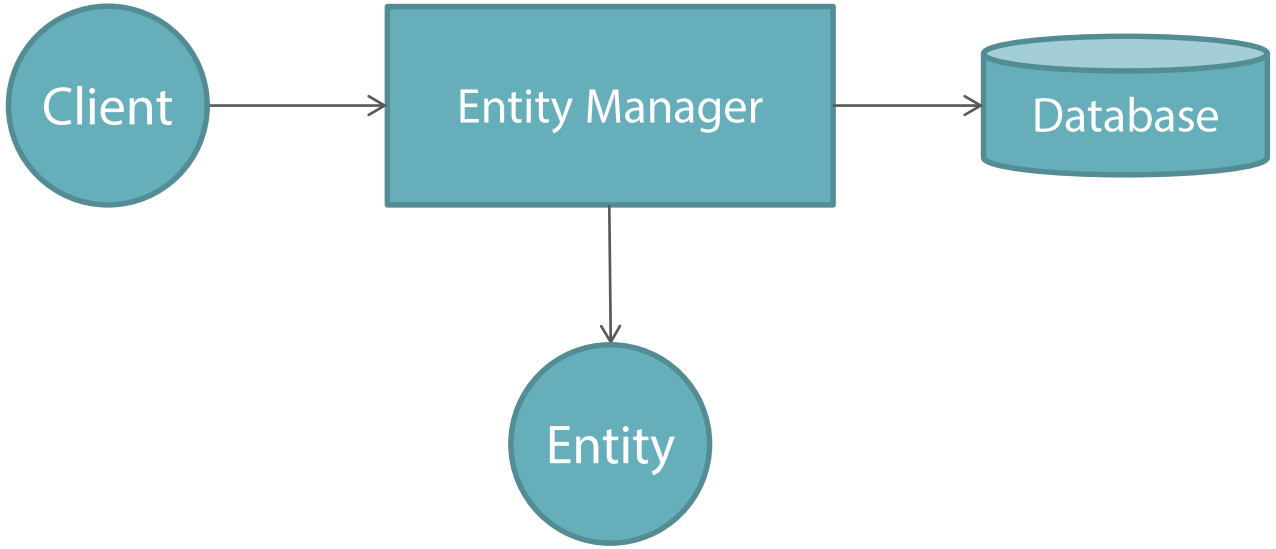
Understanding JPA

- 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

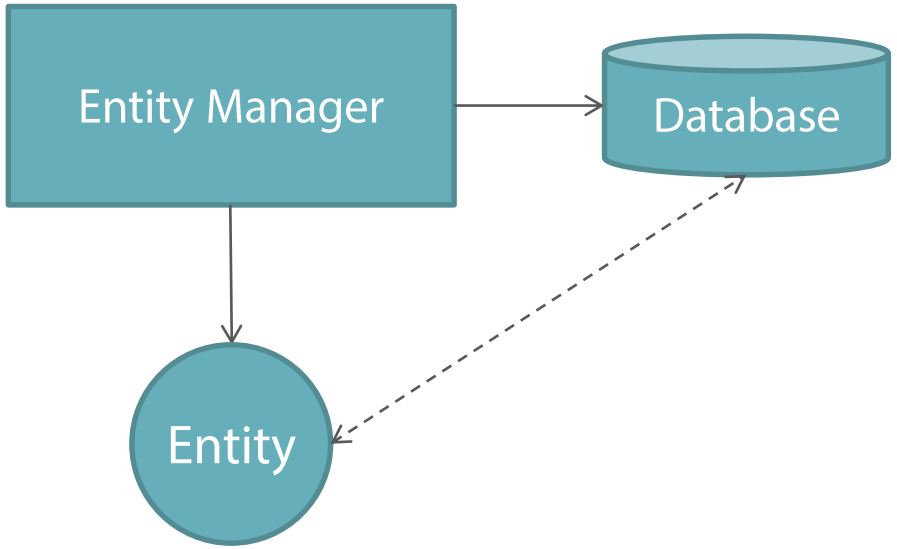
- Entity
  - Is an object
  - Live shortly in memory
  - Live persistently in database
  - Mapped to a database
  - Persistent identity
- Object
  - Just live in memory

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)elete



Entity Manager

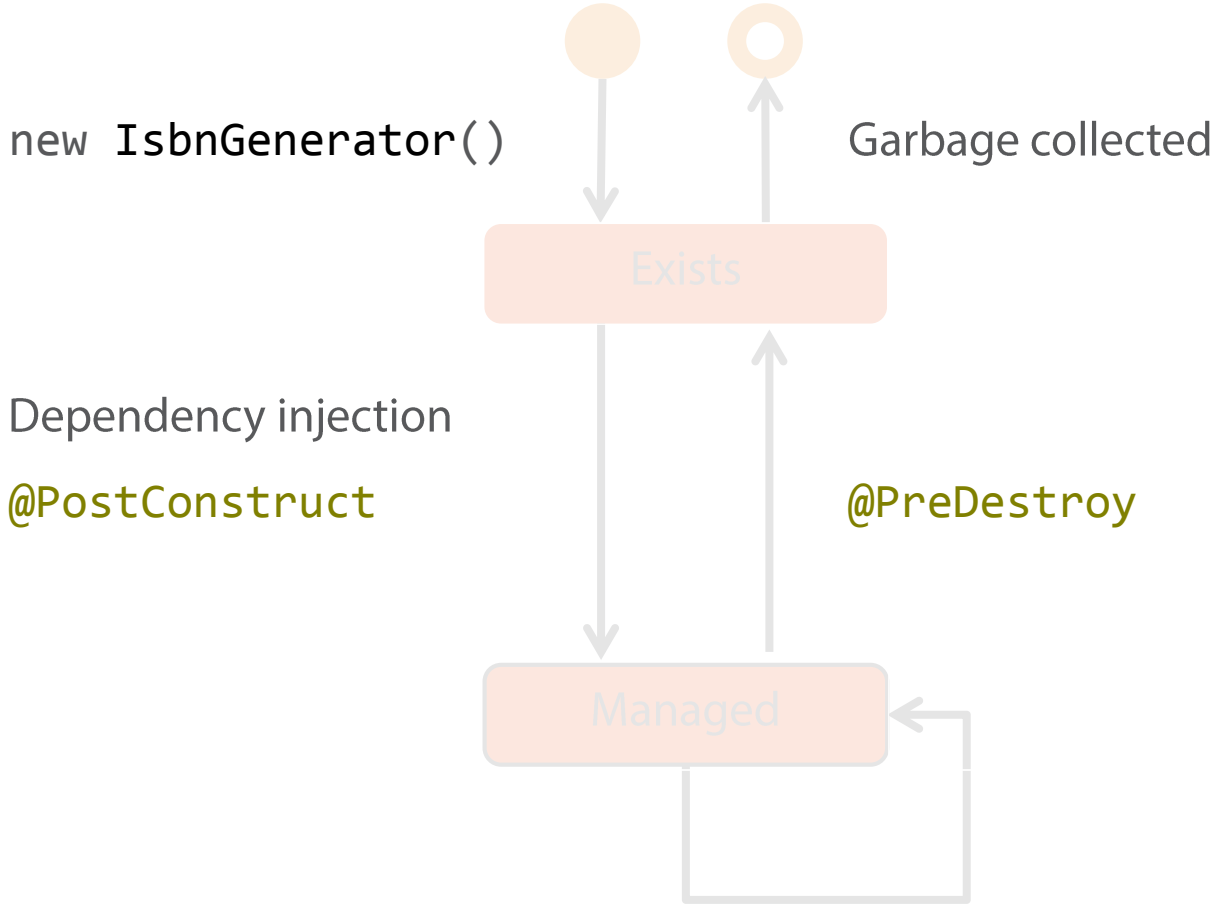
```
public class BookService {

  @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

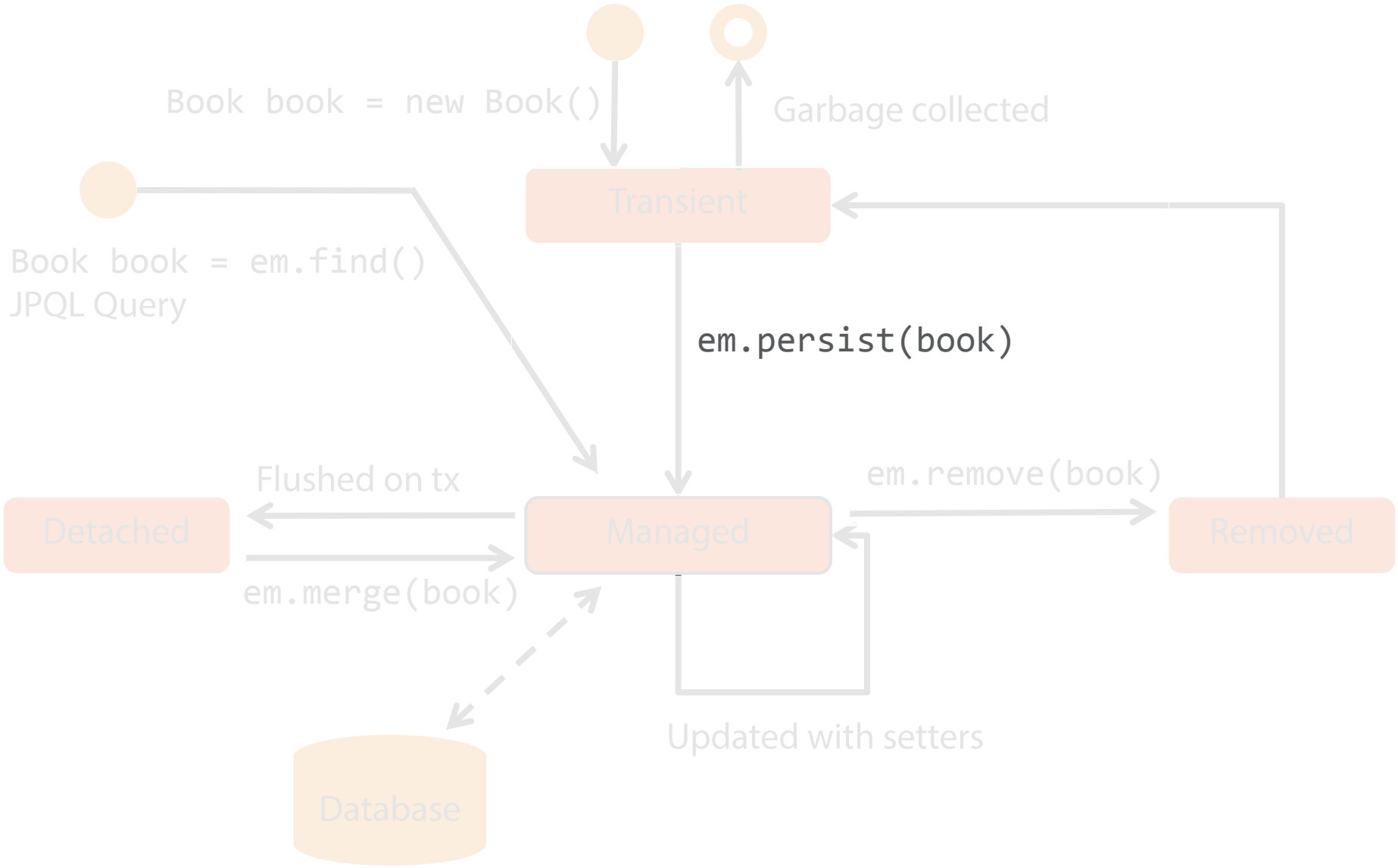
```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

<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

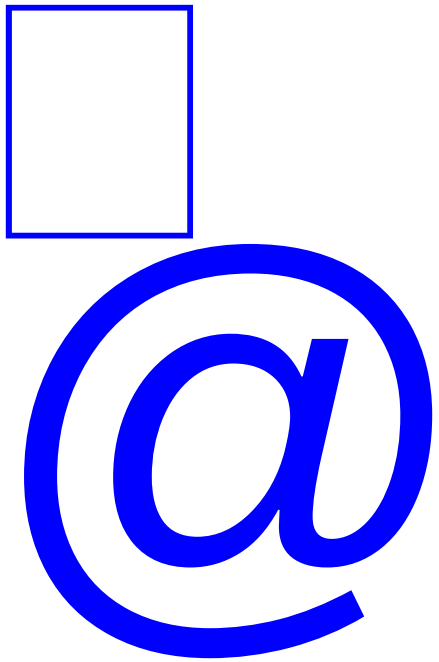
```
@Entity
public class Book {

  @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 ⇔ Table name
- Attribute name ⇔ Column name
- JDBC rules for mapping primitives
  - String ⇔ VARCHAR(255)
  - Long ⇔ BIGINT
  - Boolean ⇔ 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 EentityManager 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;
  }
}
```

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;
  }
}
```

Persistence

Entity manager

CRUD operations

JPQL queries

Transactions

Java Transaction API (JTA) 1.2

What Are Transactions?

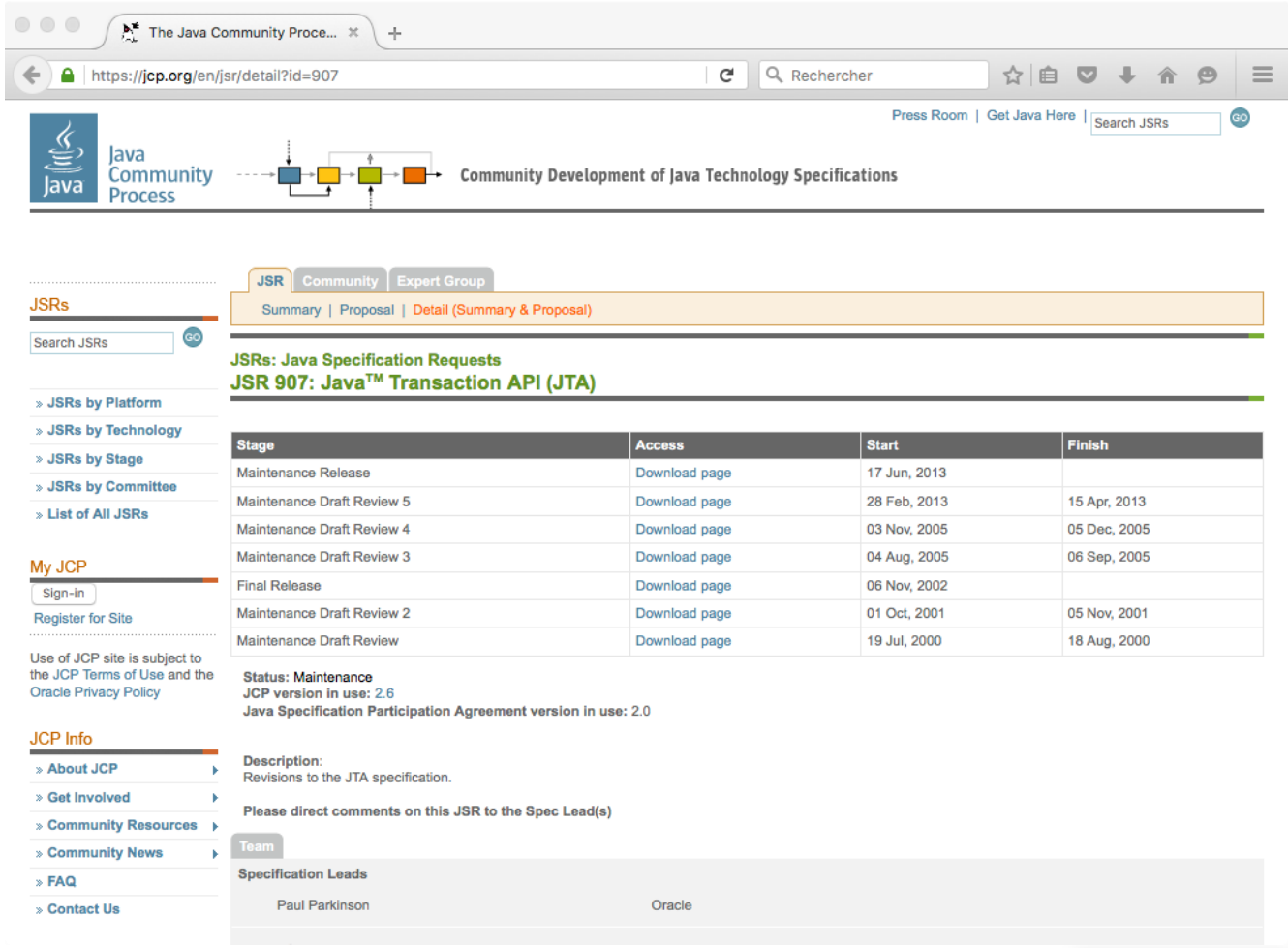
- Data is crucial
- Kept in a consistent state
- Group of operations
- Performed in a single unit of work
  - Persisting data in a database
  - Sending messages
  - Invoking web services…

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

JTA Implementations

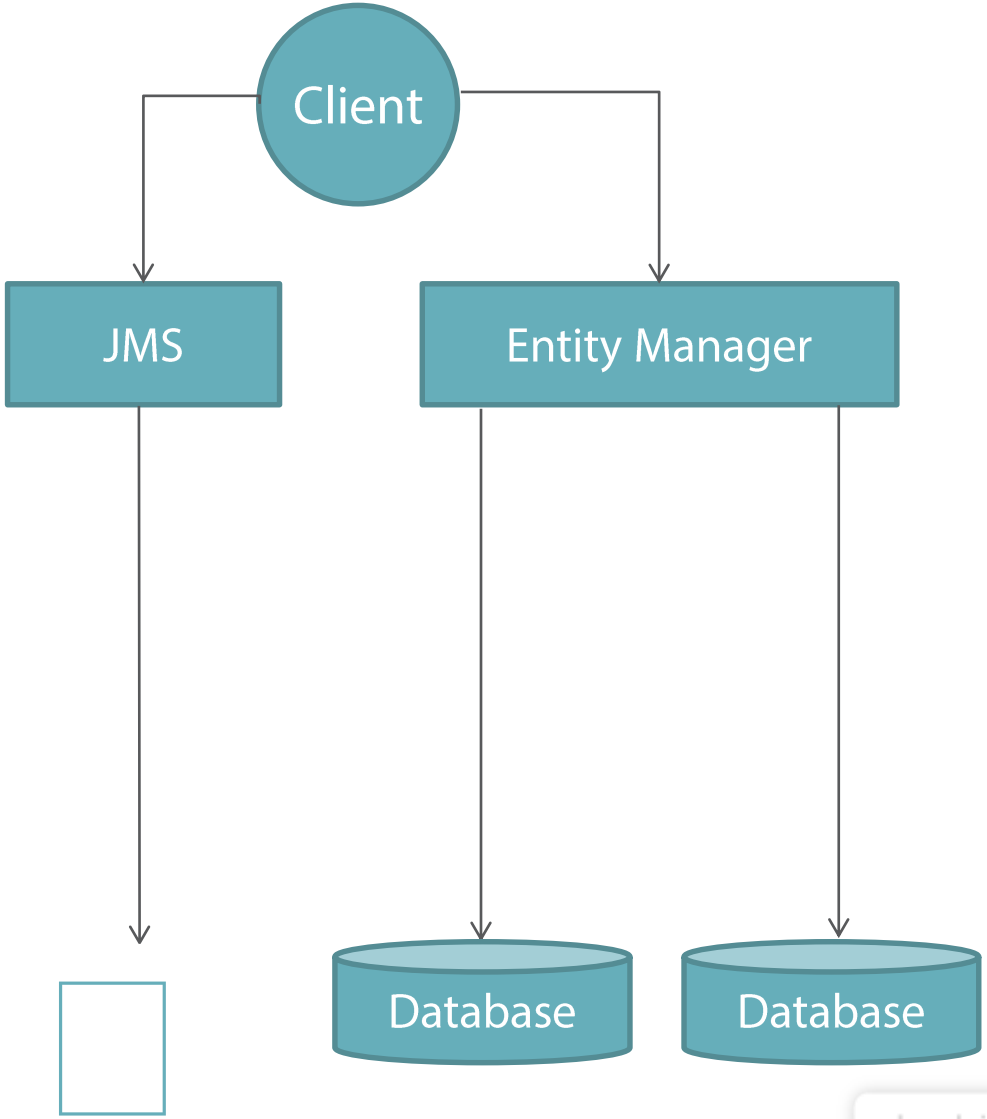
- GlassFish Transaction Manager
- JBoss Transaction Manager
- Atomikos
- Bitronix JTA

ATOMIKOS

JBoss® by Red Hat

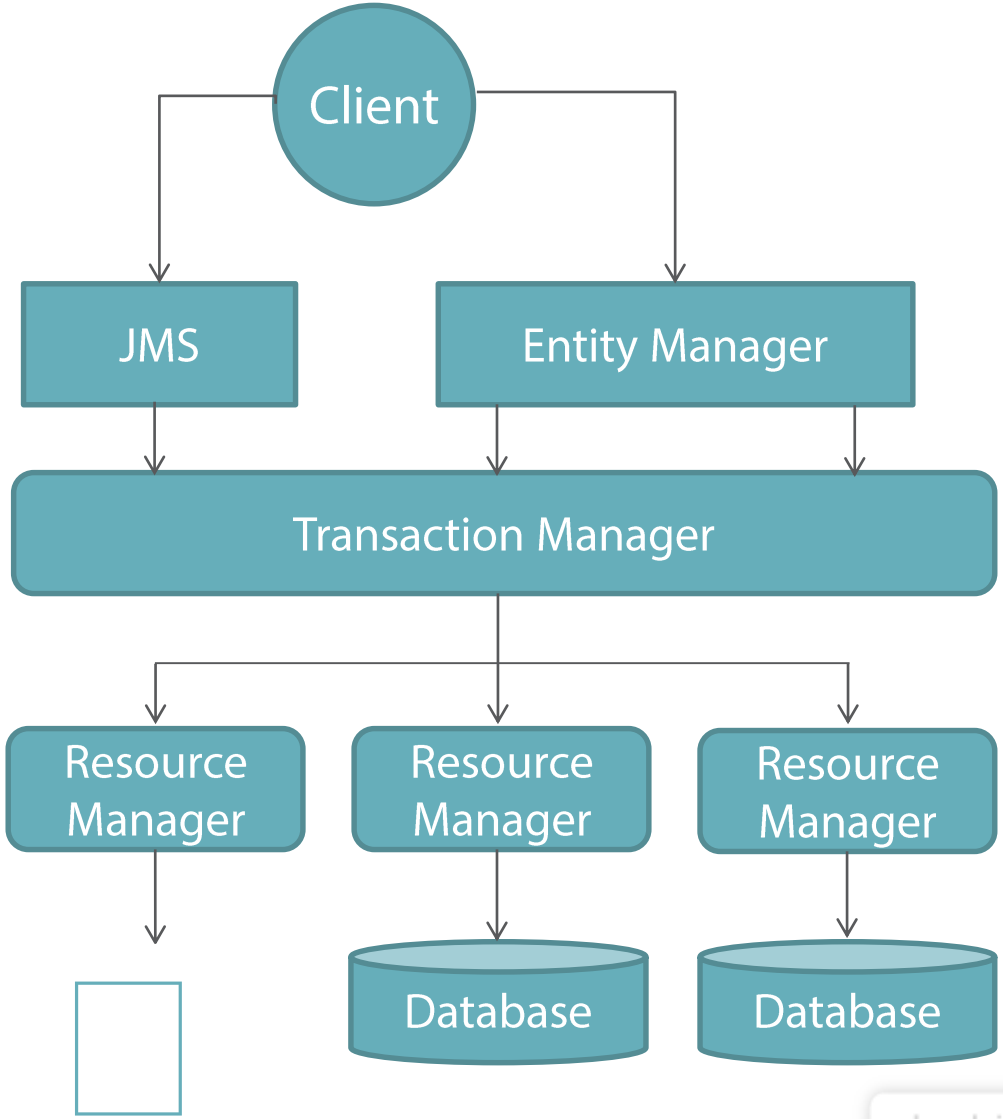
Understanding JTA

- Several databases
- Message Oriented Middleware



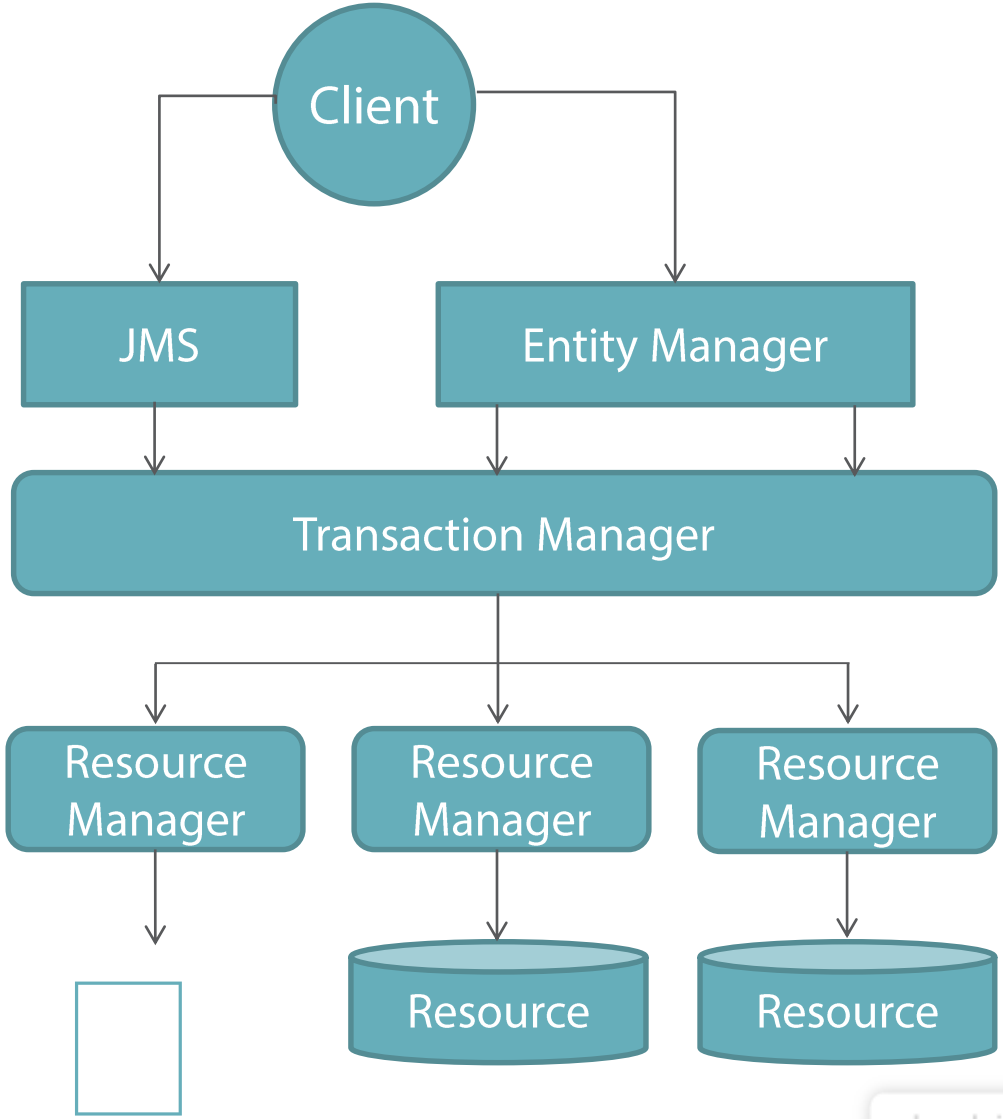
Understanding JTA

- 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
`javax.transaction`	Core JTA APIs
`javax.transaction.xa`	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

```java
@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;
  }
}
```

```java
@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

```java
@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;
  }
}
```

```java
@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();
  }
}
```

Requires New

```java
@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;
  }
}
```

```java
@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?

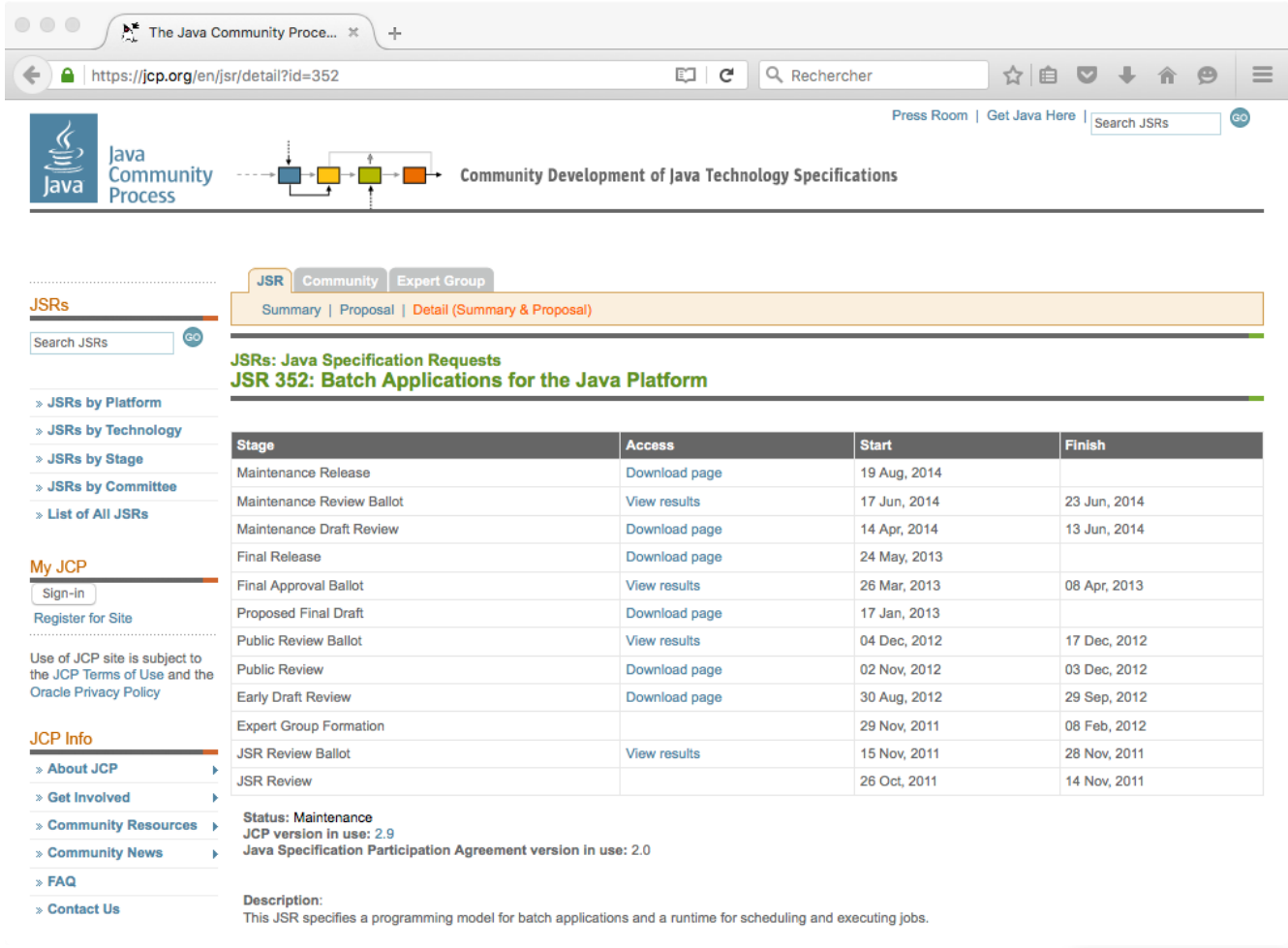
- Execution of a series of jobs
- Set of inputs
- Without manual intervention
- High-volume
- Repetitive tasks

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

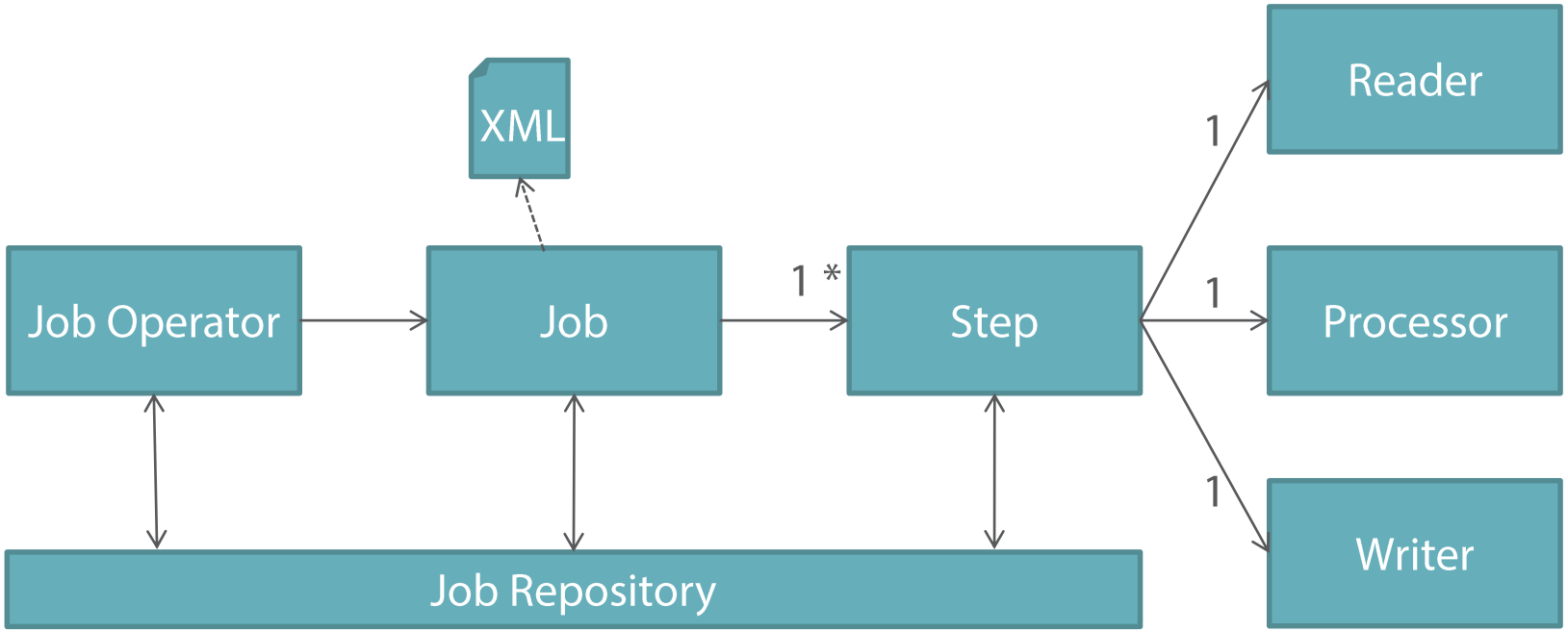
JBatch Implementations

- WebSphere Liberty Batch
- Spring Batch
- JBeret

JBeret

Understanding JBatch

- Job
- Step
  - 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

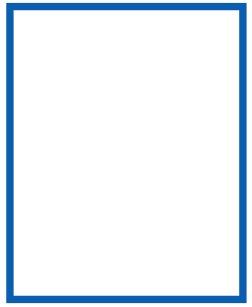
Batch Processing

Read invoices from DB

Process invoices

Write XML file

Summary



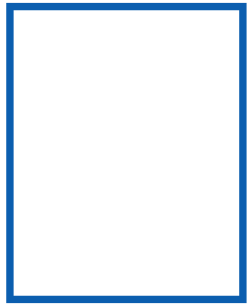
Business layer specifications

Java Persistence API

Java Transaction API

Batch processing

What's Next



Web tier

Servlets

Web pages

Web sockets