Architecture Flow

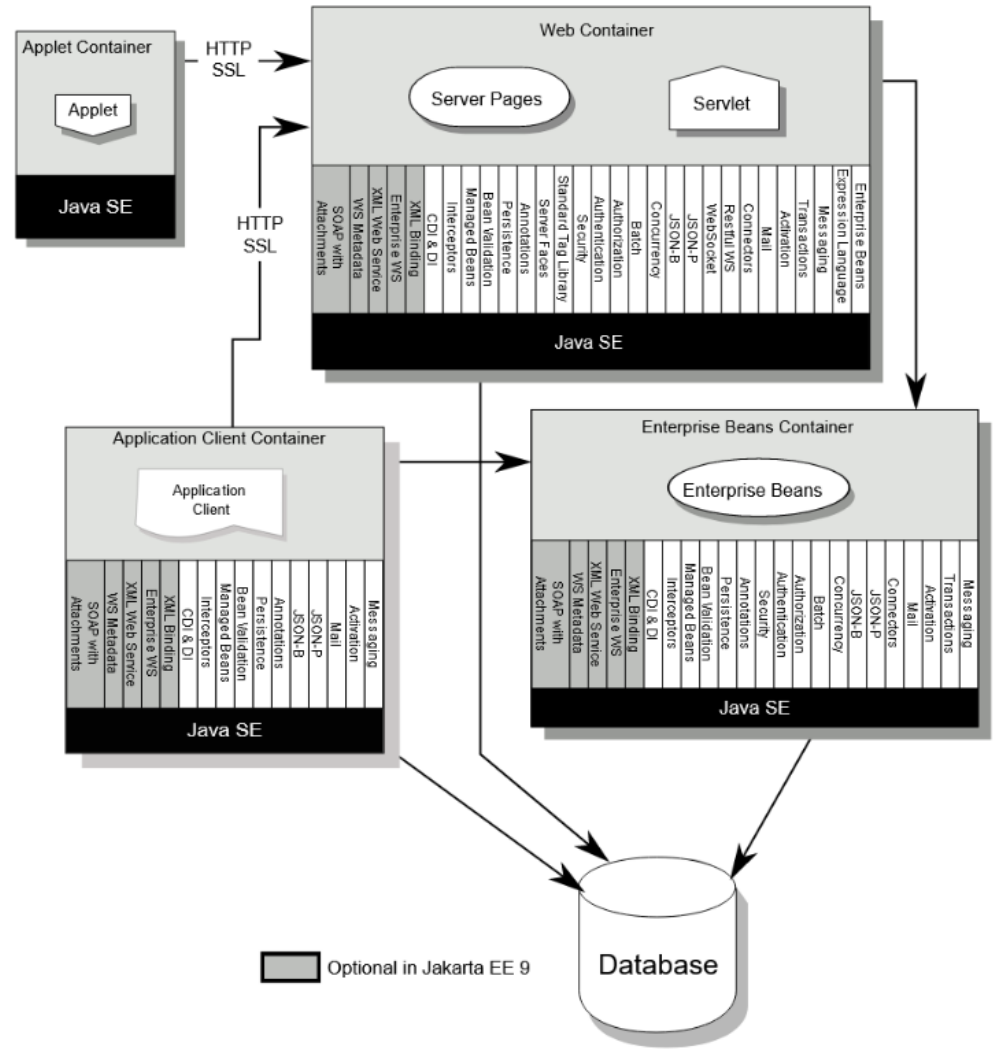


Figure 1. Jakarta EE Architecture Diagram

Client > Web Container > EJB Container > Database

Web Container - Manages the execution of web components

EJB Container - Manages the execution of enterprise bean(s) which contains business logic

Appendix

Java is an object-oriented programming language

- Everything in Java is associated with classes and objects, along with its attributes and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects
- For example: in real life, a car is an object.
- The car has attributes, such as weight and color

- And Methods, such as drive and brake

```
// Object
public class Dog {
        // attributes
  String breed;
  int age;
  String colour;

// Methods
  void barking() {
  }

  void hungry() {
  }

  void sleeping() {
  }
}
```

Getters and Setters

- For each instance variable, a getter method returns its value while a setter method sets or updates its value
- It allows you to control how important variables are accessed and updated in your code

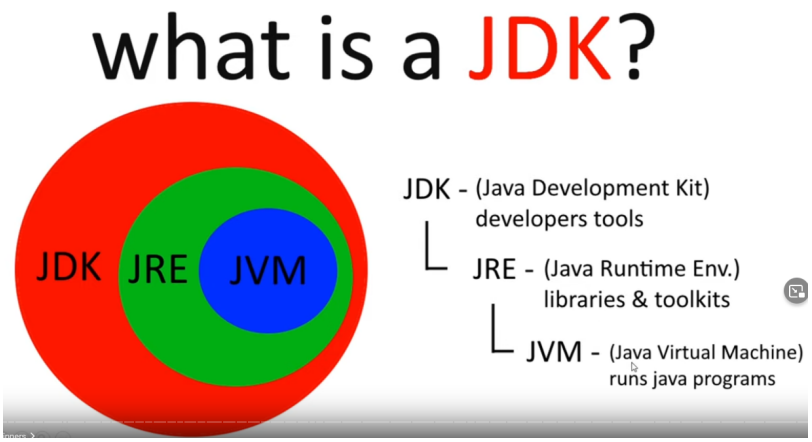
```
public class Vehicle {
  private String colour;

  // Getter
  public String getColor() {
    return colour;
  }

  // Setter
  public void setColor(String c) {
    this.color = c;
  }
}

public static void main(String[] args) {
  Vehicle v1 = new Vehicle();
  v1.setColor("Red");
  System.out.println(v1.getColor());
```

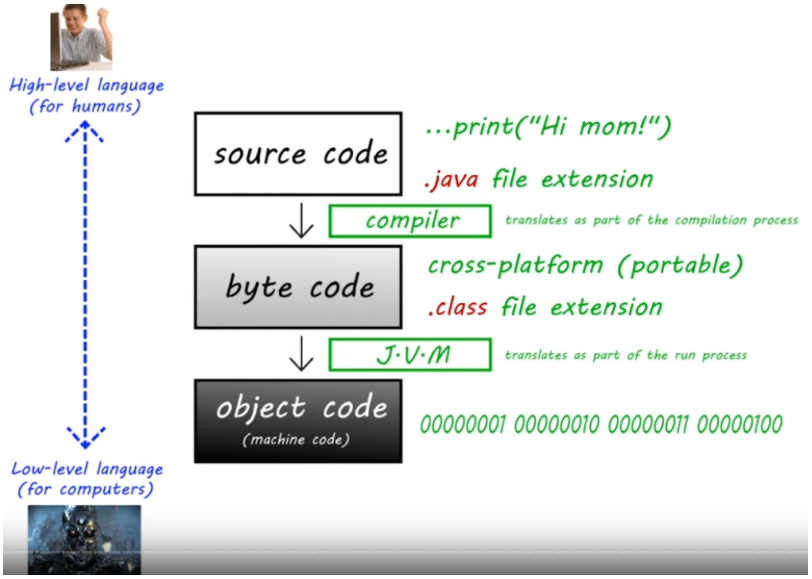
}

// Outputs "Red"

Java Annotations

1. @Override - subclass method override parent class method
2. @SuppressWarnings - suppress warnings issued by the compiler
3. @Deprecated
4. @interface - user-defined annotation
5. @Target - specify at which type, the annotation is used
6. @Retention - specify to what level annotation will be available
7. @Inherited - annotation inherited to subclass
8. @Documented - inclusion in the documentation





	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Web Container

Definitions

1. Java bean - POJOs that encapsulate many other objects into a single object so we can access this object from multiple places (Serializable, Zero-argument constructor, getter and setter methods)
2. @Named - name Java beans (e.g. insert into xhtml variables)

Servlet API

Feature: To handle the receiving of HTTP request and sending of HTTP response to and from the clients

```
@WebServlet("/ExampleServlet")
public class ExampleServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
                response.getWriter().append("Hello World");
        }

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

}
```

Asynchronous support

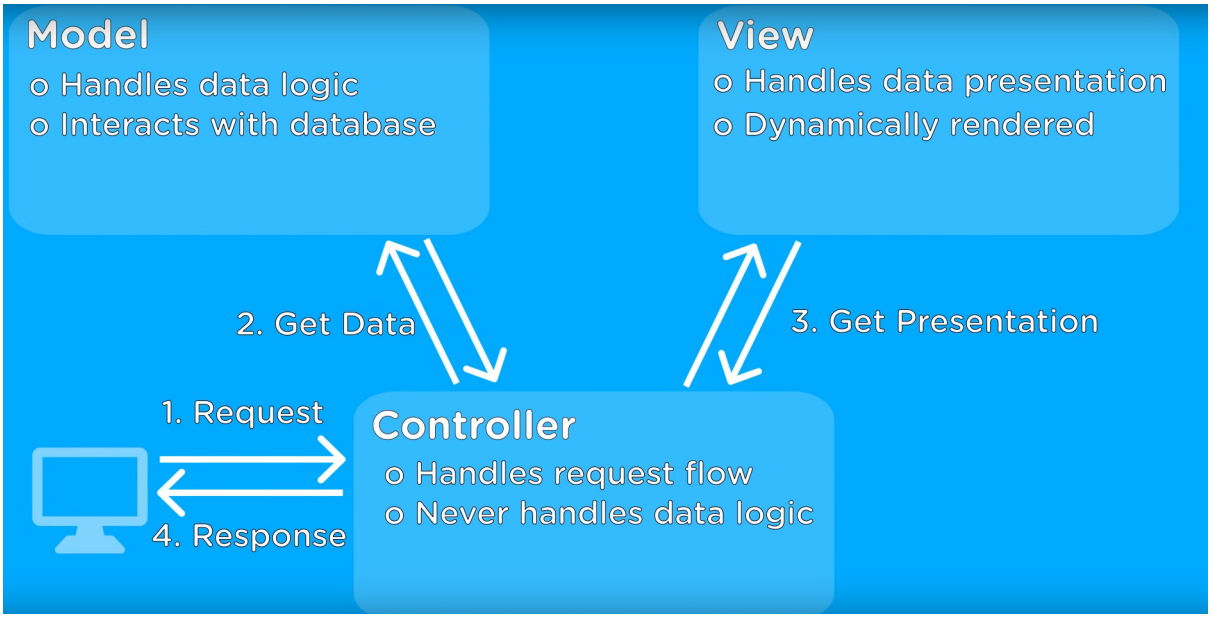
- Ensures no threads associated with requests become idle

```
@WebServlet(urlPatterns= "/CatalogServlet", asyncSupported = true)
```

```java
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    AsyncContext asyncContext = request.startAsync();

    asyncContext.start(new Runnable() {

        @Override
        public void run() {

            try {
                Thread.sleep(5000);
                System.out.println("Print the response");
                System.out.println("Reponse returned by: " + Thread.currentThread().getName());
                returnResponse(request, response);

            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }
```

Java Server Faces

- Web application framework
- Built on top of servlet API
- Based on an MVC pattern



JSF Concept

Component	Description
Faces Servlet (Controller)	Manages the request-processing lifecycle for JSF application
Facelet (View)	View featuring a component tree that corresponds to the user interface
Component (e.g PrimeFaces)	Server-side UI components corresponding to HTML components that render on the browser
Backing Bean (Model)	CDI beans with fields and methods that bind to component data and actions
Event Handlers	Methods on a backing bean bound to component actions

Jakarta Standard Tag Library

Feature: Support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags

Commonly used for XML/XHTML:

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xml:lang="en">

Expression Language

Feature: Enables users to access the data dynamically from JavaBeans components

- Immediate evaluation: ${expression}
- Deferred evaluation: #{expression}

Security API

HTTP Authentication Mechanism

Built-In Implementation

1. @BasicAuthenticationMechanismDefiniton(realmName="")
2. @FormAuthenticationMechanismDefiniton(loginToContinue=@LoginToContinue(loginPage="", errorPage=""))
3. @CustomFormAuthenticationMechanismDefiniton(loginToContinue=@LoginToContinue(loginPage="", errorPage=""))

Interface Methods

1. validateRequest() - validates an incoming request and authenticates the caller
2. secureResponse() - secures a response message
3. cleanSubject() - Clears the provided subject of principals and credentials

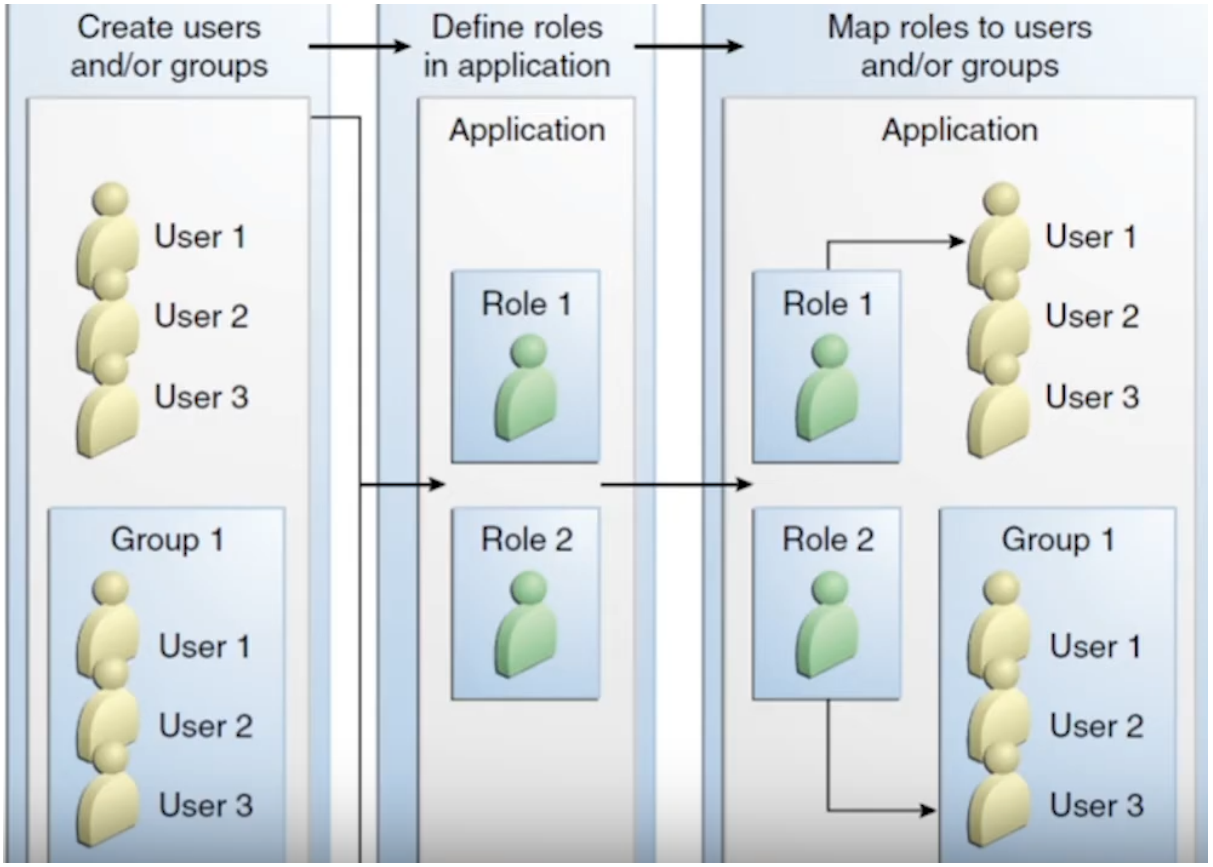
Identity Store

1. @DatabaseIdentityStoreDefinition
2. @LdapIdentityStoreDefinition
3. @EmbeddedIdentityStoreDefinition
4. Custom identity stores

Security Context API

1. AuthenticationStatus authenticate()
2. getCallerPrincipal()

Realms, Users, Groups and Roles



EJB Container

Definitions

1. Managed bean:
    - non-static, concrete class containing business logic
    - instantiated, managed, and injected by a container
    - injected into servlets, JAX-RS resources, message-driven beans, other managed beans

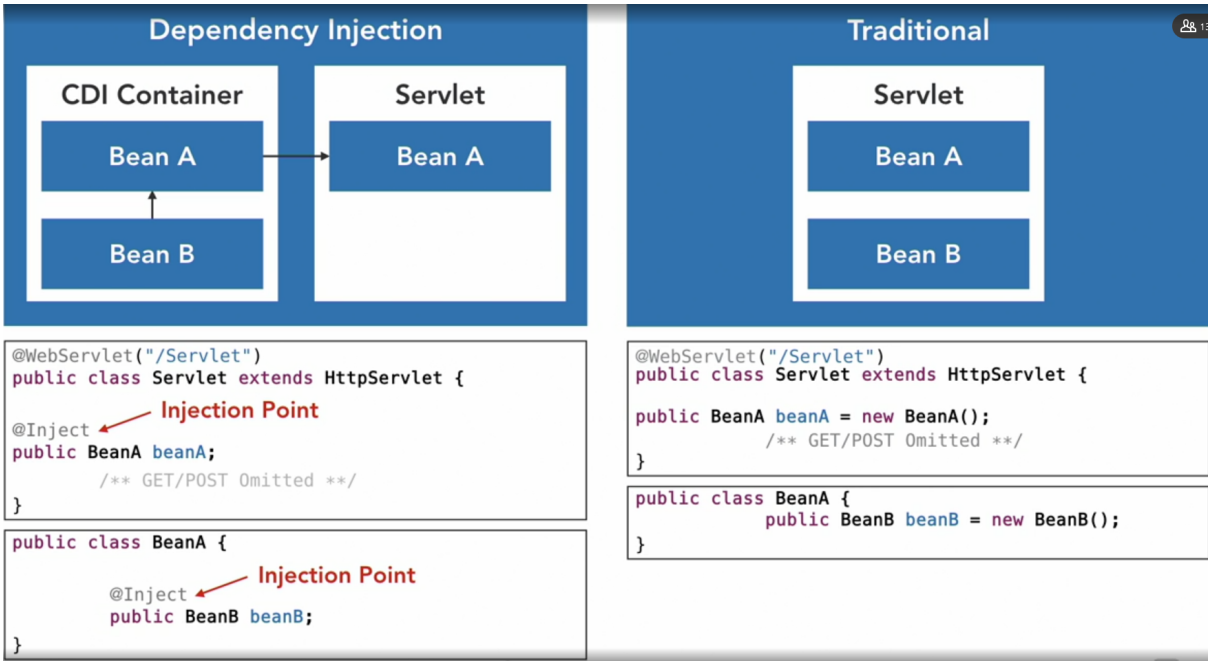
Enterprise Java Beans

- Isolates system business logic for scalability
- Benefits include: transactions, security, concurrency, networking, persistence
- Defined the type of EJBs in a file by annotations

Types of EJB

1. Session Bean - contains business logic that models an action or use case
    - @Stateful: Unique for every client and capable of maintaining the conversational state with the client or use case
    - @Stateless: Shared for every client and does not contain any state
    - @Singleton: Shared for every client and capable of maintaining state at an application level
2. Message Driven Bean - bean that processes messages sent by other components, dependent on JMS
3. Entity Bean - represent persistent data storage, user data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean

Java Context and Dependency Injection (Framework)



Overview

- Contexts - The ability to bind the lifecycle and interactions of stateful components to well-defined but extensible lifecycle contexts
- Dependency Injection - The ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject

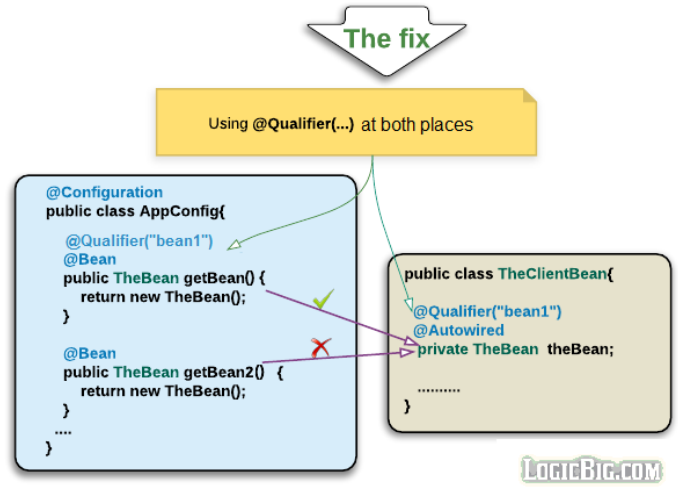
Bean Validation

Name	Annotations
Built-In Constraints	@Null @NotNull @AssertTrue @AssertFalse @Min @Max @DecimalMin @DecimalMax @Negative @NegativeOrZero @Positive @PositiveOrZero @Digits @Past @PastOrPresent @Future @FutureOrPresent @Pattern @NotEmpty @NotBlank @Email @Size

Scopes allows to associate a bean with a context which determines the visibility of the bean and helps manage its lifecycle

Qualifiers

When multiple beans have the same type, we need to specify to CDI which bean should be injected at an injection point (e.g. @Named)

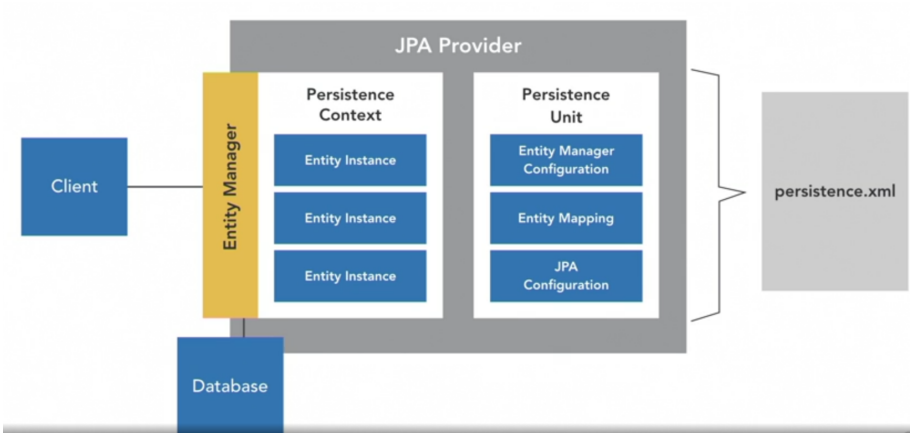



```java
public class Student {

    @NotBlank  //constraint checks if string is not null, empty or whitespace
    private String name;

    @NotNull //not null constraint (field constraint 1)
    @Email  //string constraint for email (field constraint 2)
    private String email;

    @Positive //numeric constraint for positive
    private int age;

    //constraint validates list has at least 1 but no more than 5 items
    @Size(min = 1, max = 5)
    private List<@NotEmpty String> aliases;

    @AssertTrue //boolean constraint validates value is true
    private boolean active;

    @FutureOrPresent //date constraint validates future or present
    private Date graduationDate;

}
```

DB Connection

Java Persistence API

It is used to persist data between Java objects and relational databases. JPA acts as a bridge between object-oriented domain models and relational database systems.



Entity Manager

- Manages entities within a persistence context
- Provides an API for creating, updating, deleting, and querying entities from a relational DB
- Management by container or application

Entities Annotations

Aa Annotation	Usage
Class	@Entity @Table(name="")
Field, Method	@Id @Column(name="") @GeneratedValue()
Associations	@JoinColumn @OneToOne @OneToMany @ManyToOne @ManyToMany

A domain object that represents a table in a relational database

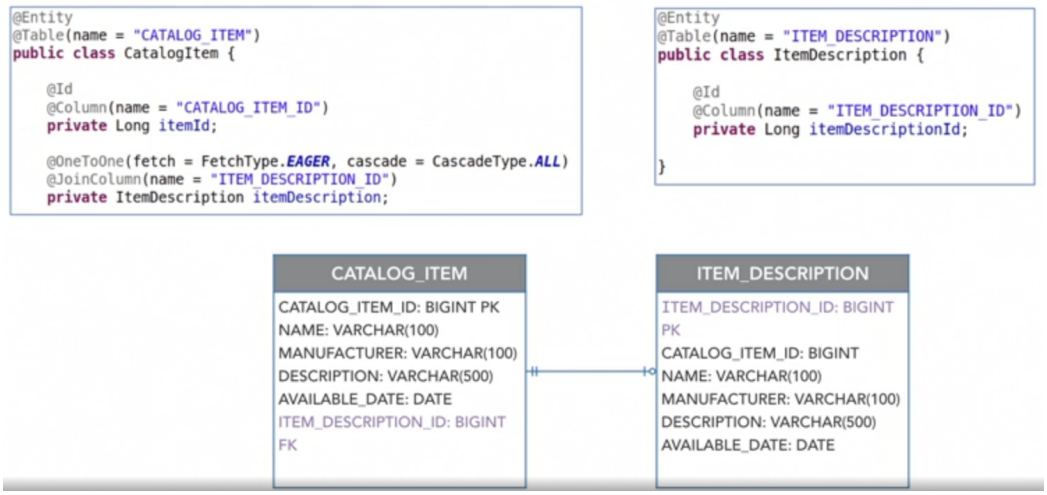
Entity instances represent a row in the particular table

Annotations map entities to relational model

3 main files to perform actions onto DB through entities :

1. Java Entity Class (Setters and Getters)
2. Persistence.xml (Map entity class and other DB configuration)
3. Java Persistence Class (To persist entity object with data, CRUD)

Entity Association (e.g. one to one):



Cascading Operations

To establish a dependency between related entities

1. @OneToOne(cascade=CascadeType.PERSIST)
2. @OneToOne(cascade=CascadeType.MERGE)
3. @OneToOne(cascade=CascadeType.DETACH)
4. @OneToOne(cascade=CascadeType.REFRESH)
5. @OneToOne(cascade=CascadeType.REMOVE)
6. @OneToOne(cascade=CascadeType.ALL)

Java Persistence Query Language Example

1. Select s.s_name from StudentEntity s

Criteria Builder API Example

EntityManager em = emf.createEntityManager();

CriteriaBuilder cb=em.getCriteriaBuilder();

CriteriaQuery<StudentEntity> cq=cb.createQuery(StudentEntity.class);

Root<StudentEntity> stud=cq.from(StudentEntity.class);

CriteriaQuery<StudentEntity> select = cq.select(stud);

Query q = em.createQuery(select);

```
List<StudentEntity> list = q.getResultList();
```