Persistence APIs

JDBC

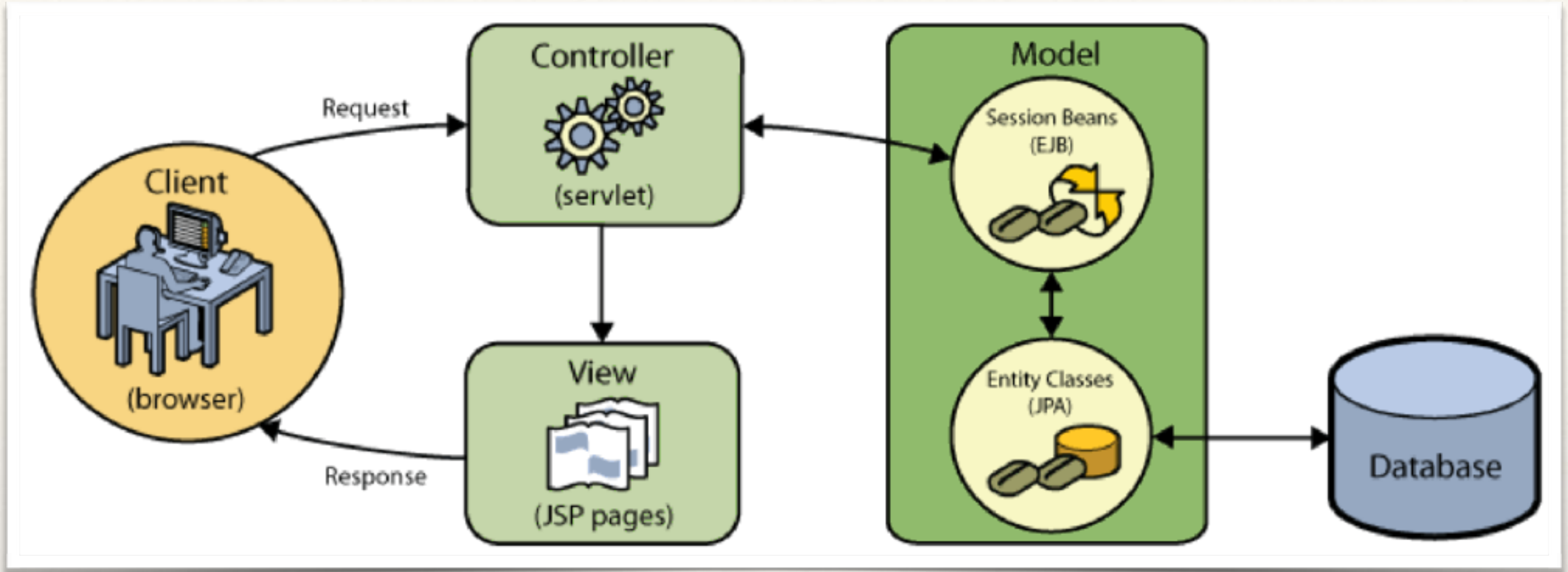
JPA

Alvaro Monge, PhD

Application Develoment

Application Develoment

- Other types of development — What happens when…
  - you hit the send button in a e-mail client?
  - you place an item in a shopping cart on e-commerce website?
  - post what you're having for dinner on Facebook?



Source: https://netbeans.org/kb/docs/javaee/ecommerce/design.html

MVC Architecture

Entity Classes use JPA to encapsulate the data in the database.

Alternatively, use JDBC.

Java packages for JDBC

- java.sql: Contains the core JDBC API.
  - API for accessing and processing data from a data source
  - Key interfaces: Connection, Driver, Statement, PreparedStatement, ResultSet
  - Key exceptions: SQLException, SQLWarning, etc.
- javax.naming: API for Java Naming and Directory Interface (JNDI)
  - Defines Context interface: set of name/object bindings
  - Easy lookup of objects by name
- javax.sql: API for server-side applications
  - DataSource, connection and statement pooling, distributed transaction, RowSet

Programming JDBC Application

- Load the JDBC driver for DB vendor
- Connect to a data source
- Execute SQL statements
    - Statement and PreparedStatement
- Process results — ResultSet
- Handle SQL errors and warnings
- Close statement and connection

Registering the JDBC Driver

- DBMS specific JDBC drivers
  - SQLite: `org.sqlite.JDBC`
  - MySQL: `com.mysql.jdbc.Driver`
  - JavaDB:
    - `org.apache.derby.jdbc.ClientDriver`
    - `org.apache.derby.jdbc.EmbeddedDriver`
- Registering the driver (loads and links the class)
  - `Class.forName(JDBC_DRIVER)`

Not needed for JDBC Type 4

StackOverflow: What is the purpose of Class.forName("JDBC_DRIVER")?

Download of SQLite JDBC Driver

DB Connection

- Use DriverManager to obtain a DB Connection

```
connection = DriverManager.getConnection(
                 JDBC_URL,
                 dbUser,
                 dbPassword);
connection.setAutoCommit(false);
```

- JDBC URL’s:
  - **SQLite**: `jdbc:sqlite:/Users/alvaro/sqlite/pa2.db`
  - **MySQL**: `jdbc:mysql://server.domain.com:3306/database`
  - **JavaDB**:
    - `jdbc:derby://server.domain.com:1527/database`
    - `jdbc:derby:/Users/alvaro/.netbeans-derby/flights`

JDBC Connection

- Create Statement objects
  - **`Statement`**: simple SQL statement (no parameters), static
  - **`PreparedStatement`**: precompiled SQL statement, parameters
  - **`CallableStatement`**: to call DB stored procedure
- Control DB transactions
  - transaction level, commit points, commit, rollback, etc.
- More…

Executing SQL

- Call one of the execute methods on `Statement` object
  - `executeQuery` — returns a ResultSet object
  - `executeUpdate` — for INSERT, UPDATE, DELETE, etc.
    - returns number of rows affected or 0
  - `execute` — if you expect more than one ResultSet object
- `PreparedStatement` (extends Statement)
  - `set`*`Type`* methods: bind SQL parameters to values, based on type
  - use it to send SQL statements as a batch

JDBC PreparedStatement

- Parameters and binding values to parameters
- Why would this be more efficient for statements that are executed many times — such as in PA 2?

JDBC ResultSet

- `ResultSet` object represents the result of a DB query
- Access data in `ResultSet` via a cursor (pointer)
  - cursor initially positioned before first row
  - next() method advances cursor to next row or returns false
- When `Statement` is created, can specify whether
  - `ResultSet` is read-only (default) or updatable
  - cursor scrolls through set only forward (default) or in both directions

Transactions

- A trasaction is a single unit of work under the ACID properties
- Auto commit

```
          connection.setAutoCommit(false);
```

- Selecting the amount of concurrency between transactions

```
connection.setTransactionIsolation(
          Connection.TRANSACTION_SERIALIZABLE);
```

- Other isolation levels:
  - REPEATABLE_READ, READ_COMMITTED, READ_UNCOMMITTED

See: Using Transactions in JDBC Basics Tutorial

Exception Handling

- `try` with resources (as of Java 7)
- SQLException
  - getSQLState()
  - getErrorCode()

For more information on try with resources, see:
http://docs.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html

Java Persistence API

Persistence using JPA

Annotating Java classes to be entities

Establishing relationships among entities

Persistence

- Relational databases
  - provide persistent storage of data
  - organized as rows in tables connected via PK/FK values
  - RDBMS guarantee ACID properties of DB transactions
- Java
  - Design and develop using object-orientation
  - Objects are transient and exist in RAM
- Software Applications
  - Object-Oriented
  - Access and manipulate information that is persistent
  - Rely on services provided by RDBMS

JDBC

Manually map rows to objects and vice versa!

ORM: Object-relational mapping

- Convert data between incompatible systems
  - RDB's: data organized as rows in tables, flat structure
  - OOP languages: composition, inheritance, etc.
- ORM gives developers an object-oriented model to transparently use entities instead of relational tables
- Programmers can create their own using JDBC
- Or you can use third-party ORM tools like:
  - Hibernate, iBATIS, JDO
  - ADO.NET Entity Framework (Microsoft)
- Java Persistence API (JPA)

JPA concepts

- **Entity**: an application-defined object that…
  - can be made persistent
  - has a persistent identity (i.e. a key)
  - is transactional (unless it's an in-memory entity)
  - is not a primitive, a primitive wrapper, or built-in object
- **Relationships**: same as relational database concept
  - Specify constraint using the annotations: @OneToOne, @OneToMany, @ManyToOne, and @ManyToMany

Example of annotations…

```
@Entity
public class Player implements Serializable {
    private String firstName;
    private String lastName;
    @ManyToOne(optional=false)
    private Team team;
    // etc...
}
```

Configuration by Exception

```
@Entity
public class Team implements Serializable {
  @Id @GeneratedValue
  private Integer id;

  @Column(name = "team_name", nullable = false, unique = true)
  private String name;

 // other attributes of Team such as teamName
  @OneToMany(mappedBy = "team")
  private Collection<Player> players;
}
```