Domain-Driven Design

SWEN-610 Foundations of Software Engineering
Department of Software Engineering
Rochester Institute of Technology
Domain driven design centers the architecture on the problem domain.

- Quote from the **DDD Community**: Domain-driven design (DDD) is an approach to developing software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts.

- The premise:
  - *Place the project’s primary focus on the core domain and domain logic*
  - *Base complex designs on a model*
  - *Initiate a creative collaboration between technical and domain experts to iteratively cut ever closer to the conceptual heart of the problem*
Let's review our project architecture.

**Client UI**
- User
- Network Connection
- HTML, CSS & JavaScript
- Any OS/HW

**Server UI**
- Controller
- View template
- Spark & FreeMarker
- Spark (for Session)
- Java Web server (Jetty)
- Any OS and HW

**Application**

**Model**

What goes in these two tiers?
DDD provides guidance for the remaining tiers.
Services provide application logic.

- The Application tier is responsible for managing the user's interaction with the application.
- It is **not** responsible for domain logic.
  - *The difference is subtle so we will restrict your use to dealing with the web session.*
- Services provide this type of logic:
  - *Manage application-wide logic and information*
  - *Maintain Model objects in the Session*
Entities provide domain logic.

- The Model tier is responsible for managing domain entities and domain logic.

- Entity responsibilities are:
  - *Process user requests/commands*
  - *Effect changes based on user requests/commands*
  - *Validate application rules*
  - *Maintain the state of the application*

- Entities are great for representing information about the world:
  - *Customers, products and orders in e-commerce*
  - *Shapes in a drawing app*
Value objects provide values for entity attributes.

- We all intuitively know what values are.
  - They are primitives in programming languages, such as int and float in Java.
  - But Java String objects are also value objects.

- Values don't change.
  - A 47 cannot be changed into a 42.
  - You cannot capitalize the first character of "fred"; instead you have to create a new string "Fred"
  - This is called "immutable"

- Lots of things can be considered values:
  - Measurements, dates, credit card numbers, money, colors, (x,y) coordinates, to name only a few
  - Java enums are natural value objects
Model objects are frequently used in collections.

- Many of the algorithms used in Model and Application components require using Entities and Value Objects in hash-based collections.

- Normal Java equality semantics are not adequate with dealing with Entities and VOs
  - An Entity must have a distinct id such that two objects with the same id must be considered equal.
  - Two Value Objects with the same data must be equal.
  - These semantic requirements imply specialized equals and hashCode methods.

- The post-class activity provides instructions on how to create these methods.
Let's review the architecture again.
This is the list of component responsibilities.

<table>
<thead>
<tr>
<th>UI Tier</th>
<th>Application Tier</th>
<th>Model Tier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Controller:</strong></td>
<td><strong>Service:</strong></td>
<td><strong>Entity:</strong></td>
</tr>
<tr>
<td>• Control the views based on the state of the application</td>
<td>• Manage application-wide logic and information</td>
<td>• Process user requests/commands</td>
</tr>
<tr>
<td>• Query the Model and Control as necessary to get information to present to the user</td>
<td>• Maintain Model objects in the Session</td>
<td>• Effect changes to the Model based on user requests/commands</td>
</tr>
<tr>
<td>• Perform simple input validation and data conversion based on input modality, e.g. String to Object</td>
<td></td>
<td>• Validate application rules</td>
</tr>
<tr>
<td>• Initiate processing of user requests/commands possibly providing data the user input</td>
<td></td>
<td>• Maintain the state of the application</td>
</tr>
<tr>
<td>• Perform data conversion for display by views</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>UI View:</strong></td>
<td></td>
<td><strong>Value Object:</strong></td>
</tr>
<tr>
<td>• Provide an interface to the user</td>
<td></td>
<td>• Provide immutable value semantics</td>
</tr>
<tr>
<td>• Present information to the user in a variety of ways</td>
<td></td>
<td>• Provide value-based logic</td>
</tr>
<tr>
<td>• Provide a mechanism for user to input data and requests</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>