Designing Spring Applications

- Previous lectures provided you with a basic understanding of what the Spring framework is and how it is used.

- This lecture, we’ll show how to go about designing a system that uses the Spring framework.

- One of the key benefits of using Spring is that it stays completely out of the way until you need it.

- Instead of thinking about how the application must fit the framework right at the start, and potentially constraining your design, you simply design your application and then add the Spring framework later!

- With Spring, you need not consider up front how clients will connect and access the application, how transactions are handled, how database connections must be managed, how authentication and authorization must be implemented, and so on. All of these concerns, having no direct relation to the application objects and the business logic, can be addressed later on with Spring.

- When you design a Spring-based system, you can start by constructing a series of interacting POJOs (plain old Java objects), forming the domain model of the system. These POJOs become the core objects in the resulting system.

- You can then use Spring’s inversion of control mechanisms (via annotations and XML-based configuration) to hook up other services to the set of POJOs
Sample Application

• To see how the Spring framework stays out of the way in your design, this lecture covers the design and initial coding of the domain model of a system called PIX.

• PIX is a Spring-based implementation of an Internet-based photo-album management system.

Overview of the PIX Album-Management System

• The example application that we work on within this lecture is a photo-album management system called PIX.

• This system is design to run on a server that is accessible via the Internet.

• The system is accessed by a variety of users and interfaces with several external systems.

• Some connections are over the Internet, while others are via local LAN or virtual private network (VPN) connections. This is representative of many real-world systems with a customer-facing Internet front-end.
High-level black-box view of the PIX system

Distinct entities that interact with the PIX system

<table>
<thead>
<tr>
<th>Entity Interacting with PIX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PixUser</td>
<td>This is a user who has registered with the PIX system and is maintaining photos on the PIX system.</td>
</tr>
<tr>
<td>Album viewer</td>
<td>This is a user of the system, but one who may not have registered with PIX. This user views the photos in one or more albums, and may have obtained a URL to the photo from a PixUser.</td>
</tr>
<tr>
<td>Affiliate</td>
<td>This is a human user who owns one or more web sites on the Internet. An affiliate registers with the PIX system and may create links from owned websites to the PIX system.</td>
</tr>
<tr>
<td>Affiliate registration</td>
<td>This is the part of the PIX system that registers affiliates.</td>
</tr>
<tr>
<td>External photo printing service</td>
<td>This is another computer system on the Internet. As a business partner of the PIX system, it provides a service: printing photos. User and photo information are sent to this service over the Internet in order for the photo to be printed.</td>
</tr>
</tbody>
</table>
The PIX System Requirements

• The following is a further exploration into some of the requirements of the system.

• These high-level requirements must be satisfied by specific features in the PIX system.

PixUser Requirements

• Before uploading photos to PIX, a user must first register for an account.
• A registered user is called a PixUser, or PixUser, in this system.
• Once registered, a user can log in and upload any number of photographs to the system.
• These photographs can be organized into albums.
• Only PixUsers logged on to the PIX system can upload photos.
• A PixUser can creatively annotate each album, and each photo within the album, using PIX.
• They can share photos with family and friends by e-mailing links to the PIX album.
• PixUsers can delete photos or albums that they own. They must be logged on before deletion is allowed.
Interactions a PixUser is involved in

- register, logon, upload photos, create albums, add comments
- view albums

PixUser

Album Viewer Requirements

- An album viewer is a user viewing photos.
- This person does not necessarily have an account with PIX.
- Album viewers who are also PixUsers can also add comments to photos — even those that don’t belong to them.
- Album viewers wishing to leave comments must first register with PIX, thus becoming PixUsers.
- An album viewer who does not register with PIX can view photos and albums but may not leave comments and/or upload photos.
Interactions an AlbumViewer is involved in

Affiliate Requirements

- The PIX system also relies on affiliates to direct traffic to the site.
- Affiliates own other web sites that link to the PIX system.
- An affiliate must first register with PIX’s affiliates registration system.
- *In practice, the affiliate would refer new users to the PIX site and be rewarded for the referral.*
- *However, this feature will not be implemented as part of the PIX system in this lecture.*
Interactions an Affiliate is involved in

External Interfacing Requirements

- Photos in the PIX system can be sent to an external photo-printing service for the creation of prints.
- The external photo-printing service runs a legacy computer system supporting interface via a message queue.
It is evident that this system is nontrivial to design and implement.

However, using the Spring framework to put things together enables you to attack the design one object-oriented layer at a time.

The first layer, and the most important one, is the domain model.

Discovering the Domain Model

• The domain model of an application can be expressed as a set of POJOs and their interactions.
• Your objective in design is to come up with a workable set of POJOs (to identify the set of programmatic objects that models the system) and enumerate their interactions (as necessitated by the application requirements).
• POJOs should represent distinct entities with behaviors and states in your system. In the case of PIX, it is natural to create POJOs around PixUsers, affiliates, pictures (photos), and albums.
• Since you do not need to track interactions by album viewers (unless they register to become PixUsers), such viewers need not be included in the domain model.
The PixUser POJO
In PIX, the PixUser is a central entity. The PixUser can upload photos, create and delete albums, and add comments to photos.

- The following information is maintained by PIX within the system:
  - First name
  - Last name
  - E-mail address
  - User name (for system login)
  - Password
  - List of albums
  - List of comments
- The user name and password are used in authentication, requiring the user to log on before accessing his photos.
- A user can create and own multiple albums on the PIX system.
- Users can create comments on each album created, and these comments are maintained by the user.

```java
public class PixUser implements Serializable {
    private String userName;
    private String firstName;
    private String lastName;
    private String email;
    private String password;

    private List<Album> albums = new ArrayList<Album>();
    private List<Comment> comments = new ArrayList<Comment>();

    public PixUser() {
    }

    public PixUser(String userName, String firstName, String lastName, String email, String password) {
        super();
        this.userName = userName;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
    }

    public List<Album> getAlbums() {
        return albums;
    }

    public void setAlbums(List<Album> albums) {
        this.albums = albums;
    }
}
```
The Affiliate POJO

- An affiliate must first register with the PIX system, and is by definition a special group of PixUsers.

- For an affiliate, the PIX system tracks the following extra information:
  - Company name
  - Fax number
  - Website URL

```java
public class Affiliate extends PixUser {
    private String companyName;
    private String faxNumber;
    private String websiteURL;

    public Affiliate() {
    }

    public Affiliate(String userName, String firstName, String lastName,
                     String email, String password, String companyName, String faxNumber, 
                     String websiteURL) {
        super(userName, firstName, lastName, email, password);
        setCompanyName(companyName);
        setFaxNumber(faxNumber);
        setWebsiteURL(websiteURL);
    }

    public String getCompanyName() {
        return companyName;
    }

    ...
```
The Picture Object

- Each Picture instance on the PIX system refers to an uploaded photograph.
- It contains information on a picture file that resides on the server.
- For each picture stored on PIX, the following information is maintained:
  - Name
  - Description
  - Size in bytes
  - Set of comments about the picture
  - Album the picture belongs in
  - Filename of the picture, as maintained on the PIX server
  - Path of the picture, as uploaded onto the PIX server
- Each picture can be in only one album.
- The filename and path to the picture are stored separately for easier manipulation; while the filename is unlikely to change, the path may change as PIX grows and needs to migrate to larger storage.

```java
public class Picture implements Serializable {
    private Integer id;
    private String name;
    private String description;
    private long size;
    private Set<Comment> comments = new HashSet<Comment>();
    private Album album;
    private String fileName;
    private String path;
    public Picture() {
    }

    public Picture(String name, String location) {
        this.name = name;
        this.path = location;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    ...
}
```
The Album POJO

- The pictures that a user uploads to the PIX system are organized as photo albums.
- Each user can group his or her photos into one or more albums.
- A particular photo in the system can belong to only one album at a time.
- A PixUser can create an unlimited number of albums.
  - Name
  - Creation date
  - User who owns the album
  - Description
  - List of pictures in the album
  - Labels for the album
- Labels are created by the PixUser who owns the album, and will be displayed alongside the icon of the album on the system.

```java
public class Album implements Serializable{
    private Integer id;
    private PixUser user;
    private String name;
    private String description;
    private Date creationDate = new Date();
    private String[] labels;
    private List<Picture> pictures = new ArrayList<Picture>();
    public Album() {
    }
    public Album(String name) {
        this.name = name;
    }
    ...
}
```
The Comment POJO

- Comments are added to pictures by album viewers.
- Since album viewers are not tracked in the PIX system, comments are not associated with any individuals.
- Instead, they are just text strings that are kept with a picture.
- A picture can contain zero or more comments.
- A PixUser can also leave comments. These comments are tracked with the PixUser.

```java
public class Comment {
    private Integer id;
    private String comment;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getComment() {
        return comment;
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    ...
}
```
The POJO Relationships

- A PixUser can have many albums
- A picture can have many comments
- A PixUser can have many comments
- An album can have many pictures
- An affiliate is a PixUser

Adding POJO Operations to Support Relationships

To maintain the relationships among the POJOs, you first need to understand the actions you need to perform. For Pix, these relationship maintenance actions include the following:

- Adding a picture to an album
- Adding an album to a PixUser
- Adding a comment to a picture
- Adding a comment to a PixUser

To support the preceding operations, you create a set of methods on the POJOs. These methods are shown in the following table.

<table>
<thead>
<tr>
<th>Action to perform</th>
<th>Associated POJO method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding a picture to an album</td>
<td>Album.addNewPicture()</td>
</tr>
<tr>
<td>Adding an album to a PixUser</td>
<td>PixUser.addAlbum()</td>
</tr>
<tr>
<td>Adding a comment to a picture</td>
<td>Picture.addComment()</td>
</tr>
<tr>
<td>Adding a comment to a PixUser</td>
<td>PixUser.addComment()</td>
</tr>
</tbody>
</table>
Establishing POJO Identity

- The identity of a Java object is determined by the equals() and hashCode() methods.
- The Java language provides adequate default implementation.
- Depending on your application, however, it may be necessary to override these methods.
- The PIX system is one in which this override is necessary.
- To complete the code for the POJOs, you need to override some of their equals() and hashCode() methods.
- If you do not override these methods, the default implementation for equals() is to use binary comparison equivalent to the following:

  ```java
  public boolean equals(Object obj) {
      return (this == obj);
  }
  ```
The Appeal of a Nonintrusive Framework

- The main appeal of a nonintrusive framework, such as Spring, is that it stays out of the way of your design and modeling activities.
- The Spring framework enables you to create your application by starting with the creation of POJOs.
- In fact, Spring enables you to keep the same POJOs throughout the design/coding cycle—typically requiring no framework-specific change to the POJO core code.
- This means that the the PIX POJOs can remain as simple as in the preceding listings, no matter if you are writing a simple GUI program using the POJO or if you are using it in a multiuser web-based server system that services tens of thousands of people.
- It is this simplicity that makes POJO-centric design one of the most intuitive means of creating an application.
- Once you identified the objects and their relationships within the system, you modeled them using the Java programming language—creating a set of interrelated POJOs. In POJO-centric design, your application is more than half complete once you have modeled the application’s POJOs.
- If you are using a conventional application framework, you must add code to the POJOs to access databases, coordinate transactions, locate services provided by the container/environment, and so on. In Spring, these tasks are performed declaratively via configuration (and via annotations), without disturbing the POJO code. This allows the POJO code to remain simple throughout the design life cycle.

There exist many benefits to keeping the POJO code simple

- It is easier to understand and maintain in the long run.
- Testing is straightforward.
- Object reuse in another application is trivial.