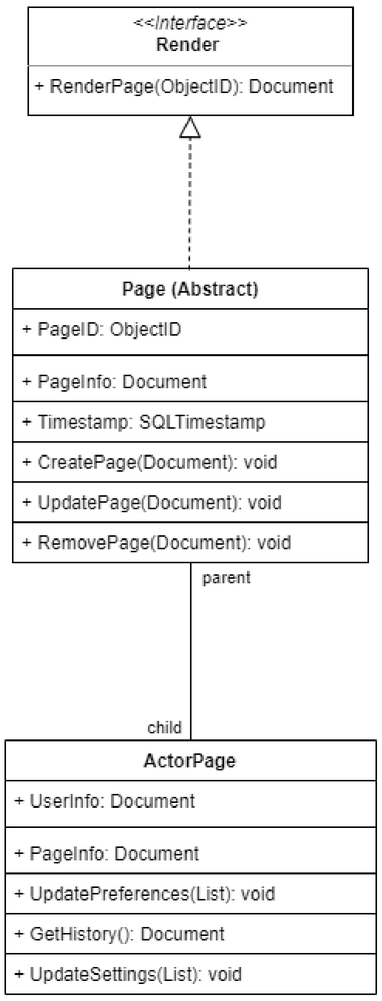
Jason C. Nucciarone

3/7/2021

M05-A02: Discover and Write an Abstract Class, Abstract Method & Interface

<u>UML Class Diagram</u>:



<<Interface>>
Render
+ RenderPage(ObjectID): Document
Page (Abstract)
+ PageID: ObjectID
+ PageInfo: Document
+ Timestamp: SQLTimestamp
+ CreatePage(Document): void
+ UpdatePage(Document): void
+ RemovePage(Document): void
parent
child
ActorPage
+ UserInfo: Document
+ PageInfo: Document
+ UpdatePreferences(List): void
+ GetHistory(): Document
+ UpdateSettings(List): void

For my design here, I decided to break up the original **BusinessPage** class into separate classes for multiple pages. My reasoning behind this split is because there are going to be multiple pages, not just one singular business page. There will need to be a page for businesses, a page for the user, etc. These page classes all share the same core methods, but they all operate slightly different. A **BusinessPage** will be different from an **ActorPage**, but they will still share common methods such as creating a page, deleting a page, and updating page. However, since all these shared methods will be implemented in very different ways for different types of pages, I decided that the abstract super-class **Page** was absolutely necessary. The **Page** super-class is used to define base abstract methods, as well as lay out a template for other pages the application will need.

As for the purpose of the **Render** interface, I implemented this interface because outside of the page classes there are still multiple classes that need to have the ability to render pages for the user. Therefore, it was necessary to implement an interface that defines the template for a ***renderPage*** method. Multiple classes will do page rendering, but they will all do it very differently depending on what part of the overall application the class controls. This interface will help keep the implementation uniform across the entire application. It would be an absolute nightmare to have multiple methods that accomplish the same goal but have very different names. It would cause an absolute headache for the development team!

The Interface:

```java
package interfaces;

import org.bson.Document;

/**
 * Interface implemented by classes that need to return a renderable
Document.
 * @author Jason C. Nucciarone
 *
 */
public interface Render {
    public Document renderPage(String id);
}
```

*I use an ObjectID method provided by MongoDB to create the ID.

The Abstract Class and Abstract Methods:

```java
package page;

// Import necessary modules
import java.sql.Timestamp;
import java.util.Calendar;

import com.mongodb.MongoClient;
import org.bson.Document;
import org.bson.types.ObjectId;

import utils.ConnectDB;
import interfaces.Render;

/**
 * Abstract parent class for Pages that need to be loaded by the application.
 * Initialize methods for interacting with created pages.
 * @author Jason C. Nucciarone
 *
 */
public abstract class Page implements Render {
    private String timestamp;
    private String pageid;
    private Document document;
    private MongoClient conn;

    // Default constructor
    public Page(){
        // Initialize variables needed by all pages
        Calendar calendar = Calendar.getInstance();
        Timestamp temp_timestamp = new
Timestamp(calendar.getTime().getTime());
        this.timestamp = temp_timestamp.toString();

        this.pageid = new ObjectId().toString();

        this.document = new Document();

        this.conn = ConnectDB.getConnection();
    }

    // Constructor for if page already exists
    public Page(String id){
        this.pageid = id;
    }

    public abstract void createPage(Document doc);

    public abstract void updatePage(Document doc);

    public abstract void removePage(Document doc);

    // Setters and getters to be used by sub-classes
    public String getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }

    public String getPageid() {
        return pageid;
    }

    public void setPageid(String pageid) {
        this.pageid = pageid;
    }

    public Document getDocument() {
        return document;
    }

    public void setDocument(Document document) {
        this.document = document;
    }

    public MongoClient getConn() {
        return conn;
    }

    public void setConn(MongoClient conn) {
        this.conn = conn;
    }
}
```

<u>The Class implementing the Interface and Abstract parent</u>:

```
package page;

// Import necessary packages
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import org.bson.Document;

import java.util.ArrayList;

/**
 * Class for managing the Actors main page. Retrieve certain information from
collection
 * in order to load dynamic UI pages for users.
 * @author Jason C. Nucciarone
 *
 */
public class ActorPage extends Page {
    private final String DatabaseName = "ACCOUNTS";
    private final String CollectionName = "ACTOR_PAGES";

    public ActorPage(){
        super();
    }

    public ActorPage(String id){
        super(id);
    }

    @Override
    public void createPage(Document doc){
        // Set page id in Document
        doc.append("page_id", getPageid());

        // Pull in Actors Document and then add settings portion to it
        doc.append("history", "");

        // Add new document to ACTOR_PAGES collection
        try {
            // Connect to database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Add the documents to the ACTOR_PAGES collection
            table.insertOne(doc);

        } catch (Exception e){
            String error = "Failed to create a page for the actor! Check
MongoDB instance.";
            System.out.println(error);
            e.printStackTrace();
        }
    }

    @Override
    public void updatePage(Document doc){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Update the document
            table.updateOne(Filters.eq("page_id", getPageid()), doc);

        } catch (Exception e){
            String error = "Failed to update the page for the actor! Check
MongoDB instance.";
            System.out.println(error);
            e.printStackTrace();
        }
    }

    @Override
    public void removePage(Document doc){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Delete document from the collection
            table.deleteOne(Filters.eq("page_id", getPageid()));
        } catch (Exception e){
            String error = "Failed to delete the page for the actor! Check
MongoDB instance.";
            System.out.println(error);
            e.printStackTrace();
        }
    }

    public Document renderPage(String id){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Grab the page
            BasicDBObject query = new BasicDBObject("page_id", getPageid());
            try (MongoCursor<Document> cursor =
table.find(query).iterator()){
                if (cursor.hasNext()){
                    setDocument(cursor.next());
                }
            }

        } catch (Exception e){
            String error = "Failed to render page! Check MongoDB instance.";
            System.out.println(error);
            e.printStackTrace();
        }

        return getDocument();
    }

    public void updatePreferences(ArrayList<String> preferences){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Update the preferences for the page in the document
            table.updateOne(Filters.eq("page_id", getPageid()),
Updates.set("preferences",
                    preferences.toArray()));

        } catch (Exception e){
            String error = "Failed to update Actor preferences! Check MongoDB
instance.";
```

```
            System.out.println(error);
            e.printStackTrace();
        }
    }

    public Document getHistory(){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Grab history from the page document
            BasicDBObject query = new BasicDBObject("page_id", getPageid());
            try (MongoCursor<Document> cursor =
table.find(query).iterator()){
                if (cursor.hasNext()){
                    setDocument(cursor.next());
                }
            }

        } catch (Exception e){
            String error = "Failed to get Actor history! Check MongoDB
instance.";
            System.out.println(error);
            e.printStackTrace();
        }

        // After getting specific Document, pull out history
        return new Document("history", getDocument().get("history"));
    }

    public void updateSettings(ArrayList<String> settings){
        try {
            // Connect to the database
            MongoClient mongoClient = getConn();
            MongoDatabase mongoDatabase =
mongoClient.getDatabase(DatabaseName);

            // Get collection
            MongoCollection<Document> table =
mongoDatabase.getCollection(CollectionName);

            // Update the settings for the page in the document
            table.updateOne(Filters.eq("page_id", getPageid()),
Updates.set("settings",
                    settings.toArray()));

        } catch (Exception e){
            String error = "Failed to update Actor settings! Check MongoDB
instance.";
            System.out.println(error);
            e.printStackTrace();
        }
    }
}
```

Other classes/files implemented to help with the construction of the class:

- ***ConnectDB*:**

This class is used to create the connection to the MongoDB instance that I am running on my machine. I implemented this class to cut down on the amount of hardcoding that I would need to do to connect to the MongoDB instance.

```
package utils;

// Import necessary modules for connecting to MongoDB instance
import com.mongodb.*;

/**
 * Basic method for connecting to database. Primarily meant for
preventing
 * the need to hardcode the path to the database.
 * @author Jason C. Nucciarone
 *
 */
public class ConnectDB {
    public static MongoClient getConnection() {
        String mongo_path = "mongodb://localhost:27017";
        testConnection(mongo_path);
        return new MongoClient(new MongoClientURI(mongo_path));
    }

    private static void testConnection(String mongo_connection_uri){
        try {
            MongoClient db = new MongoClient(new
MongoClientURI(mongo_connection_uri));

        } catch(Exception e){
            String error = "Failed to connect to MongoDB instance!
Check if MongoDB is running!";
            System.out.println(error);
            throw e;
        }
    }
}
```

- ***pom.xml*:**

This is the file I use to pull in the dependencies that I need for my Java code. I use this file to pull in the MongoDB driver and libraries as well as the BSON libraries. I also added JavaTuples because I like having tuples at my disposal.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.nucci</groupId>
    <artifactId>jason_M05_A02</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.javatuples</groupId>
            <artifactId>javatuples</artifactId>
            <version>1.2</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>bson</artifactId>
            <version>3.12.7</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongodb-driver</artifactId>
            <version>3.12.7</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongodb-driver-core</artifactId>
            <version>3.12.7</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>com.maxmind.geoip2</groupId>
            <artifactId>geoip2</artifactId>
            <version>2.15.0</version>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.6.2</version>
        </dependency>
    </dependencies>
```

```
</project>
```