Pizza Project: the APIs

The pizza1 APIs

- The most important API is the Service API. It defines what the application can do.

Doc on Service API: Has output of search for “public” in the service package, as done in class 3, notes on it, and sample service methods.

- The system also has an DAO API between the service layer and the DAO layer

Doc on DAO API: Has output of search for “public” in the dao package, notes on it, and sample DAO methods.

- The presentation layer doesn’t have an API, since it takes direction directly from the user, so we just have code snippets in Doc on Presentation Layer

Interpreting API calls

look at one from the service API:

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException

- What kind of code (what layer) calls this method?
- Answer: presentation layer

- What data does the caller need to supply in order to make this call?

Interpreting API calls

look at one from the service API:

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException

- What data does the caller need to supply in order to make this call?
  - Answer: an int room number, like 5

- What does the caller get back from this call?

Interpreting API calls

look at one from the service API:

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException

- What does the caller get back from this call?
  - Answer: a List of PizzaOrderData objects, one for each order this room has in progress (Preparing or Baked)

- Can this method throw?

Interpreting API calls

look at one from the service API:

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException

- Can this method throw?
  - Answer: Yes, a checked ServiceException (ISA Exception)

- Given that this method is an object method (not a static one) of class StudentService, what does a call look like, for room 5?

Interpreting API calls

look at one from the service API:

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException

- Given that this method is an object method (not a static one) of class StudentService, what does a call look like, for room 5?

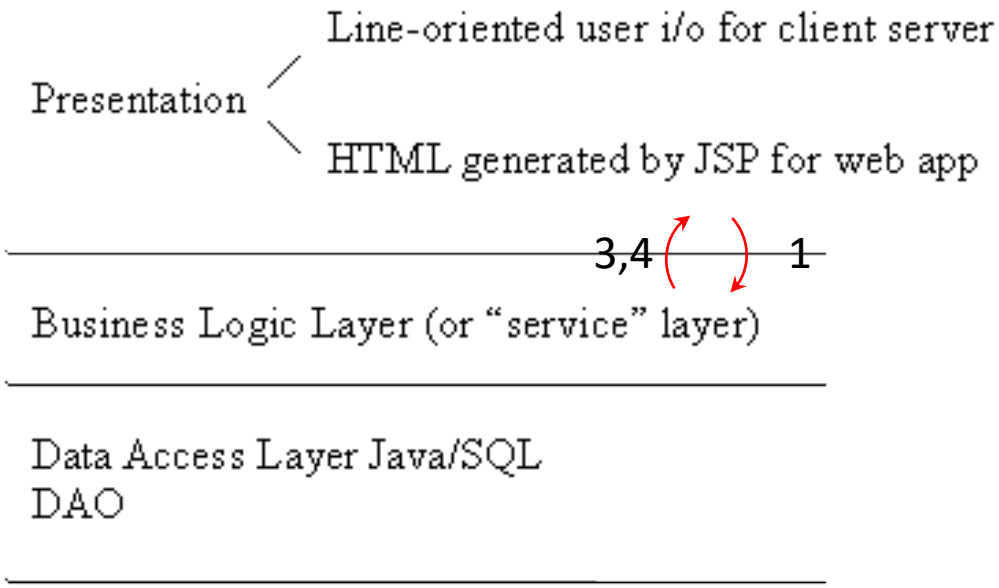
- Answer: need a ref to a StudentService object, say ss, then

```
List<PizzaOrderData> list = ss.getOrderStatus(5);
```

(in our presentation classes, that ref variable is named studentService)

Each API call has a story…

List<PizzaOrderData> getOrderStatus(int roomNumber) throws ServiceException



1. Presentation: wants order status for room 5,
Calls getOrderStatus(5)
2. Service layer gets orders for room 5 from DAO, packs them up in PizzaOrderData objects
3. Call Returns with List<PizzaOrderData> for presentation to use.
Or
4. Call throws ServiceException

Why two POJOs, PizzaOrder in domain and PizzaOrderData in service?

- We looked at PizzaOrder in domain. There is another POJO in the system, in the service package, named PizzaOrderData. We just saw it being returned by getOrderStatus().
- It's there because PizzaOrder, being mutable, shouldn't be returned to the presentation layer—that code should not be tempted to mark a pizza order « baked »--that can only happen in the service layer, the core of the app.
- PizzaOrderData has all the data from PizzaOrder, but has no mutator methods at all, so represents a PizzaOrder at a certain time, and, being invariant, can be returned to the presentation layer.
- This kind of derived object is sometimes called a **transfer object**. It doesn't deserve to be in the domain package, because it's not the working representation of the order—that's PizzaOrder. It just passively carries its data into the presentation in a safe sealed-up package.

Mutable and Invariant Domain Objects

We see that there are two kinds of domain objects:

- invariant ones like Topping that could be returned to the presentation layer as well as be involved in the core app service layer, and
- mutable ones like PizzaOrder, which should have associated invariant transfer objects like PizzaOrderData to use for providing the presentation layer with needed data.
- In fact, Topping is not returned to presentation in pizza1. Since it has only a String inside, the simple String is returned to presentation—see getToppingNames(). Simple is better.

Service Calls and how they use DAO calls

makeOrder

- It looks up current day using findCurrentDay() of AdminDAO
- It calls insertOrder(PizzaOrder order) of PizzaOrderDAO

getToppingNames: returns Topping names to presentation

- It calls findMenuToppings of MenuDAO, a DAO “finder”, returns MenuTopping objects

getOrderStatus: returns PizzaOrderData objects built from PizzaOrder objects

- It calls findOrderByRoom(- -), a DAO “finder” that returns PizzaOrder objects

DAOs are largely CRUD…

A CRUD API:
Create Retrieve Update Delete

Retrieval methods are also called finders.

Create uses insert in the DB.

Retrieve uses select, update uses update, delete uses delete

Look at Doc on DAO API again this way

FYI Since this a pretty well-defined pattern, it has useful support in, for example, Spring, which can create the method implementation by parsing the name of the method, for example

List<Person> findByLastnameIgnoreCase(String lastname); // from Spring doc on Repository

Calling down through the layers

We can see in the code:

Presentation code such as SystemTest calls `studentService.makeOrder(...)` in the service layer

Then the service code calls `pizzaOrderDAO.insertOrder(...)` in a PizzaOrderDAO

Then the DAO code calls `menuDAO.findMenuSize(sizeName)` and
Statement stmt = `connection.createStatement();`

What are these variables `studentService` and `pizzaOrderDAO`?? Also `menuDAO` and `connection`?

Looking in the sources, we see they are instance variables (or fields) of the classes SystemTest, StudentService, and PizzaOrderDAO, i.e. the classes for the various bits of code.

See next slide for the StudentService case…

Finding the instance variables (or fields) of StudentService

Here is the first part of StudentService.java:

```
package cs636.pizza.service;
import...

public class StudentService {
    private PizzaOrderDAO pizzaOrderDAO;    <---field: a ref to PizzaOrderDAO object
    private MenuDAO menuDAO;                <---field: a ref to MenuDAO object
    private AdminDAO adminDAO;              <---field: a ref to AdminDAO object

 public StudentService(PizzaOrderDAO pizzaDAO, MenuDAO mDAO, AdminDAO admDAO) {  <--constructor
        pizzaOrderDAO = pizzaDAO;   ← constructor sets fields from arguments
        menuDAO = mDAO;
        adminDAO = admDAO;
    }
…
```

So we see, for `pizzaOrderDAO.insertOrder(...)` later in this source file, inside makeOrder

^object reference to PizzaOrderDAO object named `pizzaOrderDAO` held in an instance variable of StudentService

Calling down through the layers

We just saw the method calls down through the layers of the software.

For example, in presentation code:

- studentService is an object reference to an instance of class StudentService
- makeOrder is a method of class StudentService

so `studentService.makeOrder(...)` is how you call that method of that class.

Now somewhere, sometime, the needed StudentService object was created, using the constructor we just saw code for:

```
StudentService studentService = new StudentService(...);
```

but we haven't yet seen that part. Will be covered next class (next Wednesday, after the holiday).

Note that only one instance of StudentService will be created, making it a “singleton” object.

Building and running pizza1

Recall our coverage of pom.xml for pizza1: it specifies all the needed libraries, and a plugin to build a “fat” jar.

We used **mvn clean package** to do a full build

Result we saw: **pizza1-1-jar-with-dependencies.jar** in directory target

runOnH2.sh SystemTest runs SystemTest out of that jar

runOnH2.sh:

```
# for UNIX/Linux/MacOSX
# be sure to load H2 before using this the first time
# Usage runOnH2 SystemTest|TakeOrder|ShopAdmin
java -cp target/pizza1-1-jar-with-dependencies.jar cs636.pizza.presentation.$1
```

Running the pizza1 apps using Oracle, mysql

Similarly

runOnOracle.sh SystemTest

should run SystemTest on pe07, once you have environment variables set up, as explained in DevelopmentSetup, and loaded the database, as discussed last time.

The environment variables hold your Oracle username in ORACLE_USER and password in ORACLE_PW, and part of the JDBC URL in ORACLE_SITE.

Similarly runOnOracle.sh TakeOrder runs the client program for student orders

And runOnMysql.sh ShopAdmin runs the admin app to add pizza toppings, etc.

Look at runOnOracle.sh

```
# for UNIX/Linux/MacOSX
# be sure to load Oracle before using this the first time
# Usage runOnOracle SystemTest|TakeOrder|ShopAdmin
# Depends on env vars ORACLE_SITE, ORACLE_USER, and ORACLE_PW
java -cp target/pizza1-1-jar-with-dependencies.jar cs636.pizza.presentation.$1
jdbc:oracle:thin:@${ORACLE_SITE} ${ORACLE_USER} ${ORACLE_PW}
```

For ORACLE_SITE localhost:1521:dbs3, jdbc:oracle:thin:@${ORACLE_SITE} becomes jdbc:oracle:thin:@localhost:1521:dbs3, the JDBC Url (for use with a tunnel)

We are specifying the JDBC URL, and the Oracle username and password to SystemTest (on the command line) so that it can create a JDBC connection to the database.

On pe07, ORACLE_SITE is dbs3.cs.umb.edu:1521:dbs3

By using environment variables, we can use the same script on pe07 and on our home systems.

Managing database credentials

The ORACLE_USER and ORACLE_PW are the same at home and on pe07, but the ORACLE_SITE has localhost rather than dbs3.cs.umb.edu, since at home the program needs to connect to the local end of my tunnel to Oracle.

We just set up these variables once on each system we use.

This is a well-known method of handling host-specific parameters needed by software, in development.

For production cases, the database password should be protected.

This method at least keeps passwords off the command line, where they can be seen by others via system tools like "ps -l" on Linux/Mac.

But users can be sloppy about protecting the .profile file (say) that defines the environment variables, so they may be seen by others there.

FYI: For the Amazon Cloud, see https://aws.amazon.com/secrets-manager/

Demo: running the apps

See transcript for

Fixing broken H2

Reloading H2

Running SystemTest

Running TakeOrder to order a pizza, see its status

Running ShopAdmin to make that pizza baked

Running TakeOrder again to see the pizza now baked, marking it received, then seeing its new Finished status.