A Database for Restaurants, Bars and Coffee Shops in o2

By Sonali Mathur, Nicholas Puz and Justin Boitano
CSE 132B
# Table of Contents

About the Queries ........................................................................................................... 3  
Indices .......................................................................................................................... 4  
Flow Chart of the Query Interface ............................................................................... 5  
About XML .................................................................................................................. 6  
Sample Query Results and Corresponding DTDs ...................................................... 7  
Function Prototypes .................................................................................................. 10  
OQL Function Definitions ........................................................................................... 12  
Appendix containing all information about the Classes, Design and Application ...... 23
OQL Queries Implemented

1. Search for restaurants by name.
2. Search for bars by name.
3. Search for Coffee Shops by name.
4. Search for Restaurants by cuisine.
5. Search for Restaurants by city.
6. Search for Bars by city.
7. Search for Coffee Shops by city
8. Search for Bars serving a specific drink.
9. Search for the Bar serving a specific drink with the lowest price.
10. Search for Restaurants serving items by keyword.

It is obvious why we implemented the first three basic searches. Users always need to be able to look for Restaurants, Bars and Coffee Shops by name. In the event the user needs the address, phone number, or menu of a specific restaurant this query will be extremely helpful for the user. Also, people tend to refer restaurants and bars to their friends by name. Thus, lookup by name is a quintessential service for any online Restaurant and Bar guide.

Categorization by cuisine has become the norm on Internet sites providing information about Restaurants. This is because it allows for easy navigation of a site and is a natural way to look up food. People also like to search for food specific to what mood they are in.

Our motivations for allowing lookup by city were that users don’t want to look at irrelevant information. If driving distance is a concern to the user it makes sense to limit the search to a desired area. We found that it was best to divide the area of San Diego County into cities. Also, areas like Pacific Beach house most “college” bars. Thus, it makes sense to divide the county into regions so that people can look in the areas they are most interested in.

To make our site unique and appealing to college students we chose to implement a few unique queries. After a long day of working with o2 we thought that it would be nice to search for the Bar serving our favorite drink. Also, since most college students are on a limited budget we chose to implement the search to find the Bar serving our favorite drink at the lowest price.

Lastly, we implemented a keyword search that would be especially helpful for users who want to search for a specific plate or ingredient. We feel that this feature will allow the user to bypass unwanted information and find the restaurants serving the dishes that they most enjoy. This query returns the restaurant whose menu contains the keyword that is used in the query.
Indices

create index Restaurants on Name;
create index Bars on Name;
create index CoffeeShops on Name;
create index DrinkItems on Name;
create index Restaurants on Cuisine;

We created indices on all of our persistent sets to speed lookup of Restaurants, Bars, Coffee Shops and Drinks by name. We also created indexes for restaurant cuisines because this tends to be a common way to find restaurants on the web. Also, we created indexes on DrinkItems.name because we do many frequent comparisons on its name (Bar Serving Drink, Bar with cheapest drink). We chose these as our indices because we suspect that these general queries would be the most frequent in our database. Also, these queries were generally slower than the other queries. Although indexes are supposed to speed the lookup we saw no noticeable difference in the speed of lookup after creating these indexes. We suspect that this is due to the fact that we only had about 70 bars and restaurants in our database. We would expect that the speed difference of indexing would be amplified as the number of objects in the database increase.
Flow Chart of Queries

This diagram shows the basic flow of the query menu in our application. (program Menu in application Main).
We weren't sure if some things would have been represented better as subelements or attributes. So we made the different parts of a Location are elements, and the different parts of DrinkItem are attributes. Since they are all just text strings and contain no subelements it will actually look the same to the parser (in the DOM representation) and in terms of writing XPATH queries for our database. For instance, the XPATH query to select a Bar by name is assuming a root node of Bars for the collection of bars):

```
/Bars/Bar[Name="My favorite bar"]
```

and if were were using the eXcelon XML DBMS we could have seen if the name contained "American" by the xpath query: /Bars/Bar[xln:i-contains-words(Name, 'American')]

If we were really implementing this we would have an XSL stylesheet that would go through our results and format them into a nicer interface. It would do things such as automatically displaying images (of the Restaurant, and of menu items) instead of just displaying the link. Also, by using XSL we could let the users choose their preferred layout of the results (fields shown, colors of text, etc) and we would not have to change the underlying data.
Sample Query Results and Corresponding DTDs

<!-- Result of search for bar serving a specific drink at the lowest price -->

<QUERY_RESULT>
  <BAR>Porters Pub</BAR>
  <DRINK>Coors</DRINK>
  <PRICE>2.30</PRICE>
</QUERY_RESULT>

<!-- And the corresponding DTD -->

<!ELEMENT QUERY_RESULT (BAR, DRINK, PRICE)>
<!ELEMENT BAR (#PCDATA)>
<!ELEMENT DRINK (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>

<!-- Had we used an XML Schema instead of a dtd to describe our document we would have been able to specify specific information about the datatypes involved. Such as that price is a float and must be greater than 0.0 -->

<QUERY_RESULT>
  <BAR Name="" Hours="" Picture="" Logo="" Summary="" GeneralReview="" GeneralPrice="" Atmosphere="">
    <Location>
      <Address>asdfs</Address>
      <City>cityname</City>
      <State>state</State>
      <Zip>99393</Zip>
      <Phone>858-334-3334</Phone>
      <Fax>858-334-3322</Fax>
      <Region>Costal</Region>
    </Location>
    <Food Category="Snacks">
      <ITEM name="" description="" price="" picture=""/>
      <ITEM name="" description="" price="" picture=""/>
    </Food>
    <Food Category="Burgers">
      <ITEM name="" description="" price="" picture=""/>
      <ITEM name="" description="" price="" picture=""/>
    </Food>
    <Drink Category="Beers">
      <DrinkItem name="" description="" percentalcohol="" price=""/>
      <DrinkItem name="" description="" percentalcohol="" price=""/>
    </Drink>
  </BAR>
</QUERY_RESULT>
<Drink Category="Wines">
  <DrinkItem name="" description="" percentalcohol="" price=""/>
  <DrinkItem name="" description="" percentalcohol="" price=""/>
</Drink>
</BAR>
</QUERY_RESULT>

<!-- And the corresponding DTD -->
<!ELEMENT QUERY_RESULT (BAR*) >
<!ELEMENT BAR (Location*, Food*, Drink*)>
<!ATTLIST BAR
  Name   CDATA  #REQUIRED
  Hours   CDATA  #REQUIRED
  Picture   CDATA  ''
  Logo   CDATA  ''
  Summary   CDATA  ''
  GeneralReview   CDATA  ''
  GeneralPrice   CDATA  ''
  Atmosphere   CDATA  ''>
<!Element Location (address, city, state, zip, phone, fax?, region?)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT region (#PCDATA)>
<!ELEMENT Food (ITEM+)>
<!ATTLIST Food
  Category  CDATA  #REQUIRED>
<!ELEMENT ITEM empty>
<!ATTLIST ITEM empty>
<!ATTLIST ITEM
  name   CDATA  #REQUIRED
  description   CDATA  ''
  price   CDATA  #REQUIRED
  picture   CDATA  ''>
<!ELEMENT Drink (DrinkItem+)>
<!ATTLIST Drink
  Category  CDATA  #REQUIRED>
<!ELEMENT DrinkItem empty>
<!ATTLIST DrinkItem empty>
<!ATTLIST DrinkItem
  name   CDATA  #REQUIRED
  description   CDATA  ''
  price   CDATA  #REQUIRED
  percentage alcohol   CDATA  ''>
Function Prototypes

function addRestaurant(n: string, h: string): Restaurant;
function addBar(n: string, h: string): Bar;
function addCoffeeShop(n: string, h: string): CoffeeShop;
function printRestaurants;
function printBars;
function printCoffeeShops;
function getRestaurant(n: string): Restaurant;
function getBar(n: string): Bar;
function getCoffeeShop(n: string): CoffeeShop;
function removeRestaurant(n: string): integer;
function removeBar(n: string): integer;
function removeCoffeeShop(n: string): integer;
function addR;
function addB;
function addC;
function removeR;
function removeB;
function removeC;
function modifyRestaurant;
function modifyBar;
function modifyCoffeeShop;
function mainmenu;
function coffeeShopOptions(r: CoffeeShop);
function barOptions(r: Bar);
function changePic(e: Establishment);
function changeLogo(e: Establishment);
function changeSummary(e: Establishment);
function changeGenPrice(e: Establishment);
function changeGenREV(e: Establishment);
function addLocation(e: Establishment);
function removeLocation(e: Establishment);
function changeHours(e: Establishment);
function removeApp(r: Restaurant);
function removeCourse(r: Restaurant);
function removeDes(r: Restaurant);
function removeDrink(r: Restaurant);
function addFoodItem(r: Bar);
function removeFoodItem(r: Bar);
function removeBarDrink(r: Bar);
function addBarDrink(r: Bar);
function setCSAllowStudyStatus(r: CoffeeShop);
function addReview(r: Restaurant);
function changeCuisine(r: Restaurant);
function changeAtmosphere(e: Establishment);
The above functions were all fully implemented in our database application. The first page of functions is specific to adding, removing, and modifying the objects in our database that includes Bars, Restaurants, and Coffee Shops.

The function definitions on this page are specific to the OQL queries that were implemented in our database. The names of the functions describe what the queries do. For a more complete description of what each function does look at the function definitions section below or the section on OQL queries implemented.
/*
 * query Options - this function provides options for the user interface
 * in querying the database
 */

function body queryOptions
{
    o2 integer choice;
    char inputline[100];

    while( choice != 11 )
    {
        choice = 0;
        printf("What query would you like to do?\n");
        printf("1. Search restaurants by name\n");
        printf("2. Search bars by name\n");
        printf("3. Search coffeeshops by name\n");
        printf("4. Search restaurants by type of food (cuisine)\n");
        printf("5. Search restaurants by location (city)\n");
        printf("6. Search bars by location (city)\n");
        printf("7. Search coffeeshops by location (city)\n");
        printf("8. Search bars serving a specific drink\n");
        printf("9. Search bars serving a specific drink with the lowest price\n");
        printf("10. Search restaurants inputting serving items by keyword\n");
        printf("11. Return to main menu.\n");
        gets(inputline);
        sscanf(inputline,"%d", &choice);

        switch( choice )
        {
            case(1): queryRestaurantName();
                break;
            case(2): queryBarName();
                break;
            case(3): queryCoffeeShopName();
                break;
            case(4): queryTypeOfFood();
                break;
            case(5): queryRestaurantLocation();
                break;
            case(6): queryBarLocation();
                break;
            case(7): queryCoffeeShopLocation();
                break;
            case(8): queryBarSpecificDrink();
                break;
            case(9): queryBarLowestPrice();
                break;
            case(10): queryRestaurantServingItems();
                break;
            case(11): break;
        }
    }
}
case(8): queryBarSpecificDrink();
    break;
case(9): queryBarLowestPrice();
    break;
case(10): queryRestaurantKeyword();
    break;
case(11):
    break;
default:
    printf("Error: Selection must be between 1-
11\n");
    }
}
}

/*
 * queryRestaurantName – looks for a restaurant w/ a specific name
 * in the database. If no such name exists
 * the user is notified.
 */

function body queryRestaurantName
{
    /* CHOOSE RESTAURANT BY NAME */
    o2 set(Restaurant) myresults;
    o2 string rName;
    char inputline[50];
    o2 Restaurant r;

    /* ask user for name of restaurant to search for */
    printf(" What is the name of the restaurant? \n");
    gets(inputline);
    rName = (o2 string) inputline;

    /* do the query */
    o2query( myresults,
        "SELECT r \n        FROM r in Restaurants \n        WHERE r.Name = $1", rName);

    /* now check if there were any results */
    if (count(myresults) == 0)
    {
        printf("No matches were found\n");
        printf("press ENTER to continue\n");
        gets(inputline);
        return;
    }

    /*print the results */
    for ( r in myresults)
/* CHOOSE BAR BY NAME
* if the bar name doesn’t exist the user is notified */

function body queryBarName
{
    o2 set(Bar) myresults;
    o2 string rName;
    char inputline[50];
    o2 Bar r;

    /* ask user for name of bar to search for */
    printf("What is the name of the bar? \n");
    gets(inputline);
    rName = (o2 string) inputline;

    /* do the query */
    o2query( myresults,
        "SELECT r \n        FROM r in Bars \n        WHERE r.Name = $1", rName);

    /* now check if there were any results */
    if (count(myresults) == 0)
    {
        printf("No matches were found\n");
        printf("press ENTER to continue\n");
        gets(inputline);
        return;
    }

    /* print the results */
    for ( r in myresults)
    {
        r->print;
        printf("press ENTER to continue\n");
        gets(inputline);
    }
}
/* CHOOSE COFEESHOP BY NAME
 * if the coffee shop doesn’t exist the user is notified.
 */

function body queryCoffeeShopName
{

  o2 set (CoffeeShop) myresults;
  o2 string rName;
  char inputline[50];
  o2 CoffeeShop r;

  /* ask user for name of coffeeshop to search for */
  printf(" What is the name of the coffeeshop? \n");
  gets(inputline);
  rName = (o2 string) inputline;

  /* do the query */
  o2query( myresults,
    "SELECT r \
    FROM r in CoffeeShops \
    WHERE r.Name = $1", rName);

  /* now check if there were any results */
  if (count(myresults) == 0)
  {
    printf("No matches were found\n");
    printf("press ENTER to continue\n");
    gets(inputline);
    return;
  }

  /*print the results */
  for ( r in myresults)
  {
    r->print;
    printf("press ENTER to continue\n");
    gets(inputline);
  }

};

/* RESTAURANT BY LOCATION
 * displays the contents of all restaurants in a given city
 */

function body queryRestaurantLocation
{

  o2 set (Restaurant) myresults;


o2 Restaurant r;
char inputline[50];
 printf("What is the city should the restaurant be in? \n");
 gets(inputline);
rcity = (o2 string) inputline;
rcity = "+" + rcity + "+";
 /* do the query */
o2query( myresults,
"SELECT r 
  FROM r in Restaurants, c in r.Locations 
  WHERE c.city like $1",rcity);
*/ now check if there were any results */
if (count(myresults) == 0)
{
  printf("No matches were found\n");
  printf("press ENTER to continue\n");
  gets(inputline);
}
/*print the results */
for ( r in myresults)
{
  r->print;
  printf("press ENTER to continue\n");
  gets(inputline);
  return;
}

/*/ BAR BY LOCATION
* displays the bars in a given city
*/
function body queryBarLocation
{
  o2 set(Bar) myresults;
  o2 Bar r;
  o2 string rcity;

  printf("What is the city the bar should be in ? \n");
  gets(inputline);
  rcity = (o2 string) inputline;
rcity = "*" + rcity + "+*";

/* do the query */
o2query( myresults,
    "SELECT r 
    FROM r in Bars, c in r.Locations 
    WHERE c.city like $1", rcity);

/* now check if there were any results */
if (count(myresults) == 0)
{
    printf("No matches were found\n");
    printf("press ENTER to continue\n");
    gets(inputline);
    return;
}

/*print the results */
for ( r in myresults)
{
    r->print;
    printf("press ENTER to continue\n");
    gets(inputline);
}

/* COFFEESHOP BY LOCATION
* display the coffee shops in a given city
*/
function body queryCoffeeShopLocation
{
    o2 set(CoffeeShop) myresults;
    o2 CoffeeShop r;
    o2 string rcity;
    char inputline[50];

    /* ask user to enter in location attributes */
    printf("What is the city the coffee shop should be in? \n");
    gets(inputline);
    rcity = (o2 string) inputline;
    rcity = "*" + rcity + "+*";

    /* do the query */
o2query( myresults,
        "SELECT r 
        FROM r in CoffeeShops, c in r.Locations 
        WHERE c.city like $1", rcity);

    /* now check if there were any results */
if (count(myresults) == 0) {
    printf("No matches were found\n");
    printf("press ENTER to continue\n");
    gets(inputline);
    return;
}

/*print the results */
for ( r in myresults) {
    r->print;
    printf("press ENTER to continue\n");
    gets(inputline);
}

/* FOR BARS SERVING A SPECIFIC DRINK
* displays the set of bars serving a specific drink
*/
function body queryBarSpecificDrink
{
    o2 set(Bar) myresults;
    o2 Bar b;
    o2 string bdrink;
    char inputline[50];

    /* ask user to enter in location attributes */
    printf("What is the specific drink that you want from a bar \n");
    gets(inputline);
    bdrink = (o2 string) inputline;
    bdrink = "*" + bdrink + "*";

    /* do the query */
    o2query( myresults,
        "SELECT b \n            FROM b in Bars, dc in b.Drinks, d in dc.Items\n            WHERE d.item.Name like $1",bdrink);

    /* now check if there were any results */
    if (count(myresults) == 0) {
        printf("No matches were found\n");
        printf("press ENTER to continue\n");
        gets(inputline);
        return;
    }
}
/*print the results*/
for ( b in myresults)
{
    b->print;
    printf("press ENTER to continue\n");
    gets(inputline);
}

/* FOR RESTAURANTS SERVING A SPECIFIC ITEM WITH A KEYWORD
* displays the set of restaurants containing the keyword somewhere in
* their menu.
*/
function body queryRestaurantKeyword
{
o2 set(Restaurant) myresults;
o2 Restaurant r;
o2 string rword;
char inputline[50];
/* ask user to enter in maincourse for search in description */
printf("What is the specific word that you want to search for?\n");
gets(inputline);
rword = o2 string) inputline;
rword = "*" + rword + "*";

/* do the query */
o2query(  myresults,
"SELECT r \n    FROM r in Restaurants, mccat in r.MainCourses, 
    appcat in r.Appetizers, descat in r.Desserts \n    WHERE ( exists mcitem in mccat.Items: 
                mcitem.description like $1 ) or 
        ( exists appitem in appcat.Items:
                appitem.description like $1 ) or 
        ( exists desitem in descat.Items:
                desitem.description like $1 )", 
rword);

/* now check if there were any results */
if (count(myresults) == 0)
{
    printf("No matches were found\n");
    printf("press ENTER to continue\n");
    gets(inputline);
    return;
}

/*print the results*/
for ( r in myresults)
{

r->print;
    printf("press ENTER to continue\n");
    gets(inputline);
}
}

/* SEARCH BY CUISINE IN RESTAURANT
 * displays all restaurants serving a specific cuisine
 */
function body queryTypeOfFood
{
o2 set(Restaurant) myresults;
o2 string rCuisine;
char inputline[50];
o2 Restaurant r;

/* ask user for name of restaurant to search for */
printf("What is the type of food you desire? \n");
gets(inputline);
rCuisine = (o2 string) inputline;

/* do the query */
o2query( myresults,
        "SELECT r \
          FROM r in Restaurants \
          WHERE r.Cuisine = $1", rCuisine);

/* now check if there were any results */
if (count(myresults) == 0)
{
    printf("No matches were found\n");
    printf("press ENTER to continue\n");
    gets(inputline);
    return;
}

/*print the results */
for ( r in myresults)
{
    r->print;
    printf("press ENTER to continue\n");
    gets(inputline);
}

/*
 *search for a bar with the lowest price for a given drink
 *if more than one have equally low prices the set is displayed
 */

function body queryBarLowestPrice
{
char inputline[100];
o2 string d_name;
o2 string inputstr = "";
o2 tuple(b:Bar, price:real) lowest_price;
o2 set(tuple(b:Bar, price:real)) lowest_price_set;

o2 integer number_bars_serving_drink = 0;

printf("Enter drink name to search for: ");
gets(inputline);
inputstr = (o2 string) inputline;

/* make sure the user actually entered a string */
if (inputstr == "")
{
    printf("No input was received. Please try again.\n");
    queryBarLowestPrice();
}

d_name = "\*" + inputstr + "\*";

/* now see if any bars actually serve the drink */
o2query(  number_bars_serving_drink,
           "count ( SELECT b, d.price
           \n           FROM b in Bars, dc in b.Drinks, d in
dc.Items
           \n           WHERE d.item.Name like $1",
           d_name);

if ( number_bars_serving_drink < 1 )
    printf("I'm sorry but no bars serve %s.\nI suggest that you try Quilmes, an excellent beer from Argentina.\n", inputstr);
else
{
    o2query(  lowest_price_set,
               "SELECT b, d.price \n               FROM b in Bars, dc in b.Drinks, d in dc.Items\n               WHERE d.item.Name like $1 \n               and d.price = min( SELECT d1.price\n               \n               FROM b1 in Bars, dcl in b1.Drinks, dl in
dcl.Items \n               WHERE dl.item =
               d.item)",
               d_name);

    if (count(lowest_price_set) > 1)
    {
        printf("There is a tie between the following restaurants:\n");
        for(lowest_price in lowest_price_set)
        {
            printf("\n%s sells %s for $%.2f\n", lowest_price.b->getName(), inputstr, lowest_price.price);
        }
    }
else
{
    lowest_price = element(lowest_price_set);

    printf("\n%s sells %s for $%.2f\n\n", lowest_price.b->getName(),
           inputstr, lowest_price.price);

    printf("Would you like to see more information about %s\n(enter y to see more info, and anything else to continue))?\n", lowest_price.b->getName());
    gets(inputline);
    inputstr = (o2 string) inputline;
    if (inputstr == "Y" || inputstr == "y")
        lowest_price.b->print();
}

printf("\npress ENTER to continue\n");
gets(inputline);
# Appendix

## Appendix Table of Contents:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Classes</td>
<td>24</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>26</td>
</tr>
<tr>
<td>Classes File (O2C)</td>
<td>28</td>
</tr>
<tr>
<td>Persistent Objects</td>
<td>30</td>
</tr>
<tr>
<td>Class Methods</td>
<td>31</td>
</tr>
<tr>
<td>Application Flow Diagram</td>
<td>52</td>
</tr>
<tr>
<td>Functions (Description)</td>
<td>53</td>
</tr>
<tr>
<td>Functions (Typical usage)</td>
<td>54</td>
</tr>
<tr>
<td>Functions (O2C)</td>
<td>55</td>
</tr>
</tbody>
</table>
About the Classes:

Our database is designed to hold information about Restaurants, CoffeeShops, and Bars. We created the class Establishment to hold information that is common between them, such as Address's, a name, phone number, etc. Thus both Bar and Restaurant are both subclasses of Establishment. We made CoffeeShop a subclass of bar because we realized that the CoffeeShop only has one additional attribute, AllowStudying, which specifies if the coffee shop allows studying. We did not add this to Bar because we assumed that no one would study in a Bar. But both coffee shops and bars essentially consist of a drink menu, and a food menu which is why we had CoffeeShop extend Bar.

DrinkItem:

We made DrinkItem it's own class so that the different establishments could all share the same drink items, which would be more space efficient. We did this because most establishments serve the same drinks (most restaurants serve Coke, most Bars serve Guiness, and most CoffeeShops serve Espresso), but there is a lot of overlap (restaurants serve beer, coffee, soda, etc). The point is that there is overlap AND there isn't any change in the information that needs to be represented besides the price, which is why a Drink consists of a DrinkItem and a price. We also decided to include the percent of alcohol in the drink so that we could do interesting queries such as: "at which bar can I get the highest percentage alcohol at the cheapest price." For the next revision we might add a size attribute so that we could do a more accurate calculation of the cost of the alcohol, but that seemed to be a bit extreme so we left it out of this design for simplicity. (If there is great demand perhaps it will be added ☺). The methods in drink item are there so that other methods can access the encapsulated data.

Initially we were going to do the same thing with food and have a MenuItem class that consisted of the name of the item, the description, and a the location (path) to a picture of the item. But we realized that each establishment changes the information that needs to be stored in it. Many restaurants have the same entrées with different names (which could not take advantage of menu item being a separate object unless the item didn't include the name). But the same is true for descriptions, restaurants usually prepare entrées differently (spices, side dishes, sauce) so the description also cannot be shared. And restaurants would definitely not want to share pictures of items (if they did, what motivation would a customer have for going to Restorante Piatti instead of Denny's for pasta). Hence we have eliminated every attribute from the object, so there is no reason for its existence.

Establishment

The Establishment class hold information that is found in all establishments:

- Name -- the name of the establishment.
- Hours -- the hours that the establishment is open.
- Picture -- the link (url / path) to a picture of the establishment.
- Logo -- the link (url / path) to a logo for the establishment.
- Summary -- a few sentences about the establishment.
- GeneralReview -- a general review (by a food critic preferably) of the establishment.
- GeneralPrice -- a rating of the overall price of the establishment. Ranges from 1 (inexpensive) to 5 (expensive).
- Atmosphere -- a description of the atmosphere of the establishment. Ex: classy, country, overlooking the water, etc.
- Locations: This is the set of the locations (addresses) of this establishment. This is a set so that chain restaurants which share a menu but have numerous locations only have to enter the info once.

Restaurant and Bar both extend establishment because they are establishments that offer added features (food, drinks, reviews, etc).
A restaurant is an establishment that also has the following attributes:

- **Cuisine** – a string describing the general category of food served at this restaurant (Italian, Indian, Greek).
- **Appetizers** – the set of appetizers served at the restaurant. This consists of categories and items within the categories. Possible categories include soups, salads, mini pizzas, etc.
- **Desserts** – same as above but for desserts. Possible categories include ice cream, sorbet, etc.
- **Main Courses** – same as above but for main courses. Possible categories include Steaks, Pastas, etc.
- **Drinks** – same as above but for main courses. Possible categories include sodas, smoothies, beers, wines.
- **Reviews** – a set of user reviews where users can come to the web page and give a rating to the restaurant and comment on it.

**Bar**

A bar has a general food menu (more snacks) and a drinks menu. These two menus are identical to a restaurant, but Bar’s lack the detailed food menu and user reviews. We decided not to have Restaurant be a subclass of Bar because they are two different types of establishments.

**CoffeeShop**

We thought of a coffee shop as a place that served a variety of drinks and a variety of snack type food, which is why we thought of it as a natural extension of a bar (and is why it is subclass of Bar). The only difference we’ve noticed is that some coffee shops do not allow students to sit there studying because it takes up tables. (The Living Room in downtown La Jolla is an example of such a place). So they also contain an attribute, AllowsStudying, which specifies if the CoffeeShop allows studying, and would prove extremely useful to students using our database to find study locations.
Note:

1. The above diagram references the types: Menu, MenuCategory, MenuItem, DrinkMenu, DrinkCategory, Drink, and Address. We define these just to make the code more readable (and the method more writeable). See the class definition file (next page) for their definitions.

2. Some of the methods in the above diagram are not O2C syntactically correct. Our diagramming software automatically appends () to method names even if the method takes no parameters. O2 requires that the parentheses be omitted for methods (and...
functions) with no parameters, and they are defined in this syntactically correct way in our class signatures, method bodies, and function signatures and bodies.
Classes File (O2C):

/* Type definitions: These are used to make the program more readable and writeable, but don’t add any additional functionality. */
create type MenuItem: tuple(name:string, description: string, price:real, picture:string);
create type MenuCategory: tuple (category: string, Items:set(MenuItem));
create type Menu: set( MenuCategory);
create type Drink: tuple (price: real, item: DrinkItem);
create type DrinkCategory: tuple (category: string, Items:set(Drink));
create type DrinkMenu: set(DrinkCategory);
class Address: tuple ( address: string, city: string, state: string, zip: integer, phone: string, fax: string, region: string);

class DrinkItem
type tuple
{
    Name: string,
    Description: string,
    PercentAlcohol: real
}
method
    public init (n: string, d: string, pa:real),
    public setName (n: string),
    public setDescription (d: string),
    public setPercentAlcohol(p: real),
    public getName: string,
    public getDescription: string,
    public getPercentAlcohol: real,
    public print
end;

class Establishment
type tuple
{
    Name: string,
    Hours: string,
    Picture: string,
    Logo: string,
    Summary: string,
    GeneralReview: string,
    GeneralPrice: integer,
    Atmosphere: string,
    Locations: set (Address)
}
method
    public init (n: string, h: string),
    public setName (n: string),
    public setHours (h: string),
    public setPicture (p: string),
    public setLogo (l:string),
    public setSummary (s: string),
    public setGeneralReview (gr: string),
    public setGeneralPrice (gp: integer),
    public setAtmosphere (a: string),
    public addLocation (inaddress: string, incity: string, instate: integer, phoner: string, fax: string, region: string ),
    public removeLocation(a: string, c: string, z: integer):boolean,
    public getName:string,
    public getHours:string,
    public getLogo:string,
    public getPicture:string,
    public getSummary:string,
    public getGeneralReview:string,
public getGeneralPrice:integer,
public getAtmosphere:string,
public getLocations:set(Address),
public print
end;

class Bar inherit Establishment
type tuple
{
    Drinks: DrinkMenu,
    Food: Menu
}
method
    public init(n: string, h: string),
    public addFood( cat: string, na: string, desc: string, loc:string, pr:real ),
    public addDrink( cat: string, na: string, desc: string, alc:real, pr:real ),
    public removeFood( cat: string, name: string):boolean,
    public removeDrink( cat: string, name: string):boolean,
    public print
end;

class CoffeeShop inherit Bar
type tuple
{
    AllowStudying: boolean
}
method
    public init(n: string, h:string),
    public setAllowStudying(a: boolean),
    public getAllowStudying:boolean,
    public print
end;

class Restaurant inherit Establishment
type tuple
{
    Cuisine: string,
    Appetizers: Menu,
    MainCourses: Menu,
    Desserts: Menu,
    Drinks: DrinkMenu,
    Reviews: set( tuple( name: string, rating: integer, comment: string))
}
method
    public init (n: string, h: string),
    public addAppetizer( cat: string, na: string, desc: string, loc:string, pr:real ),
    public addMainCourse( cat: string, na: string, desc: string, loc:string, pr:real ),
    public addDesert( cat: string, na: string, desc: string, loc:string, pr:real ),
    public addDrink( cat: string, na: string, desc: string, alc:real, pr:real ),
    public addReview( name: string, na: string, rating: integer, comment: string),
    public removeAppetizer( cat: string, name: string):boolean,
    public removeMainCourse( cat: string, name: string):boolean,
    public removeDesert( cat: string, name: string):boolean,
    public removeDrink( cat: string, name: string):boolean,
    public removeReview( name: string):boolean,
    public getCuisine:string,
public setCuisine(c:string),
public print

end;
**Persistent Objects:**

We used the following named objects to keep information entered into the restaurant database persistent.

```plaintext
name DrinkItems:set(DrinkItem);
name Restaurants:set(Restaurant);
name Bars:set(Bar);
name CoffeeShops:set(CoffeeShop);
```
Class Methods:
(note: descriptions are given in the code)

selectAllClass

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DrinkItem Methods ~~~~~~~~~~~~~~~~*/

/********************************************************
* DrinkItem - init
* this is the constructor for the DrinkItem class.
* args:
*   n: string - name of the drink
*   d: string - description of the drink
*   pa:real  - percentage of alcohol in the drink.
*********************************************************/
method body init (n: string, d: string , pa: real) in class DrinkItem
{
    self -> Name = n;
    self -> Description = d;
    self->PercentAlcohol = pa;
}

/********************************************************
* DrinkItem - setName
* this sets the name of the drink
* args:
*   n: string - name of the drink
*********************************************************/
method body setName (n: string) in class DrinkItem
{
    self -> Name = n;
}

/********************************************************
* DrinkItem - setDescription
* this sets the description of the drink
* args:
*   d: string - description of the drink
*********************************************************/
method body setDescription (d: string) in class DrinkItem
{
    self -> Description = d;
}

/********************************************************
* DrinkItem - setPercentAlcohol
* this sets the percentage alcohol of the drink
* args:
*   p: real - percent of alcohol in the drink Ex: .40 for Absolut Vodka
*********************************************************/
method body setPercentAlcohol (p: real) in class DrinkItem
{
    self->PercentAlcohol = p;
}
method body print in class DrinkItem
{
    printf("DrinkItem:\n");
    printf("\tName: %s\n", self->Name);
    printf("\tDescription: %s\n", self->Description);
    printf("\tPer cent Alcohol: %f\n", self->PercentAlcohol);
};

/method body getName:string in class DrinkItem
{
    return self->Name;
};

/method body getDescription:string in class DrinkItem
{
    return self->Description;
};

/method body getPercentAlcohol: real in class DrinkItem
{
    return self->PercentAlcohol;
};

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Establishment Methods
~~~~~~~~~~~~~~~~~~~~~~~*/

/method body init in class Establishment
{
* args:
*   n: string - name of the establishment
*   d: string - the hours that establishment is open.
*********************************************************
method body init(n: string, h:string) in class Establishment
{
    self->Name = n;
    self->Hours = h;
};

/*********************************************************
* Establishment - setName
* this sets the name of the establishment
* args:
*   n: string - name of the establishment
*********************************************************/
method body setName(n: string) in class Establishment
{
    self->Name = n;
};

/*********************************************************
* Establishment - setHours
* this sets the hours of the establishment
* args:
*   h: string - hours that the establishment is open.
*********************************************************/
method body setHours(h: string) in class Establishment
{
    self->Hours = h;
};

/*********************************************************
* Establishment - setPicture
* this sets the location of the image that is the Picture for this establishment.
* args:
*   p: string - location of the picture of the establishment
*********************************************************/
method body setPicture(p: string) in class Establishment
{
    self->Picture = p;
};

/*********************************************************
* Establishment - setLogo
* this sets the location of the image that is the logo of this establishment.
* args:
*   l: string - location of the logo of the establishment
*********************************************************/
method body setLogo(l: string) in class Establishment
{
    self->Logo = l;
}
/* Establishment - setSummary

* this sets the summary of the establishment, which
* should be a paragraph or so describing the
* establishment
*
* args:
* s: string - summary of the establishment
******************************************************************************/
method body setSummary(s: string) in class Establishment
{
    self->Summary = s;
};

/*********************** *********************************
* Establishment - setGeneralReview
*
* this sets the general review of the establishment
*
* args:
* gr: string - general review of
* establishment
******************************************************************************/
method body setGeneralReview(gr: string) in class Establishment
{
    self->GeneralReview = gr;
};

/********************************************************
* Establishment - setGeneralPrice
*
* this sets the general price of the establishment
*
* args:
* gp: integer - general price of
* establishment
******************************************************************************/
method body setGeneralPrice(gp: integer) in class Establishment
{
    self->GeneralPrice = gp;
};

/********************************************************
* Establishment - setAtmosphere
*
* this sets the atmosphere of the establishment
*
* args:
* a: string - atmosphere of establishment
******************************************************************************/
method body setAtmosphere(a: string) in class Establishment
{
    self->Atmosphere = a;
};

/********************************************************
* Establishment - getName
*
* this gets the name of the establishment
*
* returns:
* string - name of the establishment
******************************************************************************/
method body getName: string in class Establishment
{ return self->Name;
};

******************************************************************************
* Establishment - getHours
* this gets the hours of the establishment
* returns:
* string - hours that the establishment
* is open.
******************************************************************************
method body getHours:string in class Establishment
{
 return self->Hours;
};

******************************************************************************
* Establishment - getPicture
* this gets the location of the image that is the Picture
* for this establishment.
* returns:
* string - location of the picture of
* the establishment
******************************************************************************
method body getPicture: string in class Establishment
{
 return self->Picture;
};

******************************************************************************
* Establishment - getLogo
* this gets the location of the image that is the logo
* of this establishment.
* returns:
* string - location of the logo of
* the establishment
******************************************************************************
method body getLogo: string in class Establishment
{
 return self->Logo;
};

******************************************************************************
* Establishment - getSummary
* this gets the summary of the establishment, which
* should be a paragraph or so describing the
* establishment
* returns:
* string - summary of the establishment
******************************************************************************
method body getSummary: string in class Establishment
{
 return self->Summary;
};

******************************************************************************
* Establishment - getGeneralReview
* this gets the general review of the establishment
* returns:
  * string - general review of
  * establishment
***********************************************************/
method body getGeneralReview: string in class Establishment
{
  return self->GeneralReview;
};

/***********************************************************/
* Establishment - getGeneralPrice
* this gets the general price of the establishment
* returns:
  * integer - general price of
  * establishment
***********************************************************/
method body getGeneralPrice:integer in class Establishment
{
  return self->GeneralPrice;
};

/***********************************************************/
* Establishment - getAtmosphere
* this gets the atmosphere of the establishment
* returns:
  * a: string - atmosphere of establishment
***********************************************************/
method body getAtmosphere: string in class Establishment
{
  return self->Atmosphere;
};

/***********************************************************/
* Establishment - getLocations
* this gets the atmosphere of the establishment
* returns:
  * set(Address) - the set of locations
  * of this establishment
***********************************************************/
method body getLocations:set(Address) in class Establishment
{
  return self->Locations;
};

/***********************************************************/
* Establishment - addLocation
* this adds a location to the set of addresses for this
* establishment
* args:
  * inaddress: string - the street address of this location
  * incity: string - the city of this location
  * instate: string - the state of this location
  * inzip: integer - the zip code of this location
  * inphone: string - the phone number of this location
  * infax: string - the fax number of this location
inregion: string – the region/neighborhood that this location is in.

**************************************************************************************************
method body addLocation (inaddress: string, incity: string, instate: string, inzip: integer, inphone: string, infax: string, inregion: string)

in class Establishment
{
    o2 Address loc;
    loc.address = inaddress;
    loc.city = incity;
    loc.state = instate;
    loc.zip = inzip;
    loc.phone = inphone;
    loc.fax = infax;
    loc.region = inregion;
    self->Locations += set(loc);
};

Remark - removeLocation

* this removes the location with the given address, city, and street from the establishment

* args:
  * a: string – the street address of the location to remove
  * c: string – the city that the location to remove is in
  * z: integer – the zip code of the location to remove.

* returns:
  * string – true -> the given location was found an removed.
  * false -> the given location did not exist
  * (was not a location for this establishment)

**************************************************************************************************
method body removeLocation(a: string, c: string, z: integer):boolean in class Establishment
{
    o2 Address loc;
    o2 boolean result = false;
    for (loc in self->Locations where (loc.address == a && loc.city == c && loc.zip == z))
    {
        self->Locations -= set(loc);
        result = true;
    }
    return result;
};

Remark - print

* this method prints all of the information that is contained within the current establishment.

**************************************************************************************************
method body print in class Establishment
{
    o2 tuple (address: string, city: string, state: string, zip: integer, phone: string, fax: string, region: string) loc;
    printf("\tName: %s\n", self->Name);
    printf("\tHours: %s\n", self->Hours);
    printf("\tPicture: %s\n", self->Picture);
    printf("\tLogo: %s\n", self->Logo);
    printf("\tSummary: %s\n", self->Summary);
    printf("\tGeneralReview: %s\n", self->GeneralReview);
    printf("\tGeneralPrice: %d\n", self->GeneralPrice);
    printf("\tAtmosphere: %s\n", self->Atmosphere);
    printf("\tLocations:\n");
    for (loc in self->Locations)
{  printf("\t\tAddress: %s\n", loc.address);
  printf("\t\tCity: %s\n", loc.city);
  printf("\t\tState: %s\n", loc.state);
  printf("\t\tZip: %d\n", loc.zip);
  printf("\t\tPhone: %s\n", loc.phone);
  printf("\t\tFax: %s\n", loc.fax);
  printf("\t\tRegion: %s\n", loc.region);
}

/~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~ Coffee Shop
~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

method body init(n:string, h:string) in class CoffeeShop
{
  self->Bar@init(n, h);
}

method body setAllowStudying(a:boolean) in class CoffeeShop
{
  self->AllowStudying = a;
}

method body getAllowStudying:boolean in class CoffeeShop
{
  return self->AllowStudying;
}

method body print
{
  //...
* this method prints all of the information that is
  * contained within the current CoffeeShop
  * It calls Bar's print method to print its attributes
  * that are extended from Bar
  *******************************************************/
method body print in class CoffeeShop
{
  self->Bar@print;
  printf("\tAllowStudying: %d\n", self->AllowStudying);
};

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Bar~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/***************************************************************/
* Bar - init
* this is the constructor for the class Restaurant. Since
* Restaurant derives Establishment we call the Establishments
* constructor.
* args:
*  n: string - name of the Restaurant
*  h: string - hours of operation
  *******************************************************/
method body init(n:string, h:string) in class Bar
{
  self->Establishment@init(n,h);
};

/***************************************************************/
* Bar - print
* this method prints all of the information that is
* contained within the current Bar.
* It calls Establishment's print method to print its attributes
* that are extended from Establishment
  *******************************************************/
method body print in class Bar
{
  o2 Drink d;
  o2 DrinkCategory dc;
  o2 MenuItem mi;
  o2 MenuCategory mc;
  self->Establishment@print;
  printf("\tDrinks:\n");
  for (dc in self->Drinks)
  {
    printf("\t\t\t%s\n", dc.category);
    for (d in dc.Items)
    {
      printf("\t\t\t\t%s\n", d.item->getName());
      printf("\t\t\t\t\t%d\n", d.item->getDescription());
      printf("\t\t\t\t\t%\n", d.price);
    }
  }
  printf("\tFood:\n");
  for (mc in self->Food)
  {
    printf("\t\t\t\t%s\n", mc.category);
    for (mi in mc.Items)
    {
  
```c
printf("\t\t\t\t%s\n", mi.name);
printf("\t\t\t\t%s\n", mi.description);
printf("\t\t\t\t$%.2f\n", mi.price);
printf("\t\t\t\t%s\n", mi.picture);
}
);

/*********************************************************
* Bar - addFood
*
* this method adds an appetizer to the set of appetizers
* that are available at the current restaurant.
* if the category is already defined the appetizer is added
* to the existing set or else a new category is created
*
* args:
*   cat: string - category of the appetizer
*   na: string - name of the appetizer
*   desc: string - description of the appetizer
*   loc: string - location of a picture of the appetizer
*   pr: real - the price of the appetizer
**********************************************************/
method body addFood ( cat: string, na: string, desc: string, loc:string, pr:real ) in class Bar
{
    o2 MenuCategory curr_category;
    o2 MenuItem new_MenuItem;
    o2 MenuItem curr_MenuItem;
    /* build the menu item we want to add */
    new_MenuItem.name = na;
    new_MenuItem.description = desc;
    new_MenuItem.price = pr;
    new_MenuItem.picture = loc;
    /* see if category already exists */
    for( curr_category in self->Food)
    {
        /* if this is true the category already exists in the appetizers */
        set */
        if( curr_category.category == cat )
        {
            self->Food -=set(curr_category);
            curr_category.Items += set(new_MenuItem);
            self->Food +=set(curr_category);
            return;
        } /* if no category exists we fall through to here */
        /* now we must create a new category and add the */
        /* new menu item to it */
        self->Food += set(tupple(category: cat,Items:set(new_MenuItem)));
    }
}

/*********************************************************
* Bar - addDrink
*
* this method adds a drink to the set of drinks
* that are available at the current restaurant.
* to minimize the amount of information we must persist
```
* we have a global set called DrinkItems. This set contains all drinks served at every restaurant. Thus, when we are adding a new Drink to the restaurant we first see if it exists in this global set and add the reference from this set to the set of Drinks in the current object. 
* also, for each restaurant we must check to see if the category of the drink is already defined. If it is the drink is added to the existing set or else a new set for the category is created.

```
**args:**
  * cat: string - category of the drink
  * na: string - name of the drink
  * desc: string - description of the drink
  * loc: string - location of a picture of the drink
  * pr: real - the price of the drink

******************

```method body addDrink( cat: string, na: string, desc: string, alc: real, pr: real)
in class Bar
{
  o2 DrinkItem new_item = new DrinkItem(na, desc, alc);
  o2 Drink addDrink = tuple( price: pr, item: new_item);
  o2 DrinkCategory curr_category;
  o2 DrinkItem curr_item;
  /* determine if drink already exists in global scope */
  for(curr_item in DrinkItems)
  { /* if it already exists add this item */
    if( curr_item == addDrink.item)
    {
      for( curr_category in self->Drinks)
      { /* if this is true the category already exists in the set */
        if( curr_category.category == cat )
        {
          self->Drinks+=set(curr_category);
          curr_category.Items += set(tuple(price: pr, item: curr_item));
          self->Drinks+=set(curr_category);
          return;
        }
      }
    } /* if no category exists we fall through to here */
    /* now we must create a new category and add the new menuItem to it */
    self->Drinks +=set(tuple(category: cat, Items:set(tuple(price: pr, item: curr_item))));
    return;
  }
}
/* if we reach this point then the Drink Item doesn't exist yet */
DrinkItems += set(addDrink.item);
for( curr_category in self->Drinks)
{
  /* if this is true the category already exists in the set */
  if( curr_category.category == cat )
  {
    self->Drinks+=set(curr_category);
    curr_category.Items += set(addDrink);
    self->Drinks+=set(curr_category);
    return;
  }
}
/* if no category exists we fall through to here
 * now we must create a new category and add the
 * new menuItem to it
 */
self->Drinks += set(tuple(category: cat, Items: set(addDrink)));

/*********************************************************
* Bar - removeDrink
*
* this method removes the drink from the set of drinks
* for the Bar.
*
* args:
*  cat: string - category of the drink
*  name: string - name of the drink
*
* return:
*  true - if drink is found and removed
*  false - if the drink is not found
***********************************************************/
method body removeDrink ( cat: string, name: string): boolean in class Bar
{
    o2 DrinkCategory curr_category;
    o2 Drink curr_drink;
    for (curr_category in self->Drinks)
    {
        for (curr_drink in curr_category.Items where
            curr_category.category == cat)
        {
            if (curr_drink.item->getName == name)
            {
                self->Drinks -= set(curr_category);
                curr_category.Items -= set(curr_drink);
                if (count(curr_category.Items) > 0)
                    self->Drinks += set(curr_category);
                return true;
            }
        }
    }
    return false;
}

/*********************************************************
* Bar - removeFood
*
* this method removes the appetizer from the set of appetizer
* for the Food.
*
* args:
*  cat: string - category of the appetizer
*  name: string - name of the appetizer
*
* return:
*  true - if Food is found and removed
*  false - if the Food is not found
***********************************************************/
method body removeFood ( cat: string, name: string): boolean in class Bar
{
    o2 MenuCategory curr_category;
    o2 MenuItem curr_food;
    for (curr_category in self->Food)
    {
        if (curr_category.category == cat)
            if (curr_food->getName == name)
            {
                self->Food -= set(curr_category);
                curr_category.Items -= set(curr_food);
                if (count(curr_category.Items) > 0)
                    self->Food += set(curr_category);
                return true;
            }
    }
    return false;
}
for( curr_food in curr_category.Items where curr_category.category == cat) {
    if( curr_food.name == name ) {
        self->Food = set(curr_category);
        curr_category.Items = set(curr_food);
        if(count(curr_category.Items) > 0 )
            self->Food += set(curr_category);
        return true;
    }
}
return false;
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Restaurant  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/********************************************************
* Restaurant - setCuisine
*
* this sets the Cuisine of a restaurant
* (ex: "Italian", "French", etc).
*
* args:
*  c: string - name of the Cuisine
*********************************************************/
method body setCuisine(c:string) in class Restaurant {
    self->Cuisine = c;
}

/********************************************************
* Restaurant - getCuisine
*
* this sets the Cuisine of a restaurant
* (ex: "Italian", "French", etc).
*
* returns:
*  string - name of the Cuisine
*********************************************************/
method body getCuisine:string in class Restaurant {
    return self->Cuisine;
}

/********************************************************
* Restaurant - init
*
* this is the constructor for the class Restaurant. Since
* Restaurant derives Establishment we call the Establishments
* constructor.
*
* args:
*  n: string - name of the Restaurant
*  h: string - hours of operation
*********************************************************/
method body init ( n: string, h: string) in class Restaurant {
    self->Establishment@init(n, h);
}

/********************************************************
* Restaurant - addAppetizer
*
* this method adds an appetizer to the set of appetizers
* that are available at the current restaurant.
* if the category is already defined the appetizer is added
* to the existing set or else a new category is created
*
*  args:
*   cat: string - category of the appetizer
*   na: string - name of the appetizer
*   desc: string - description of the appetizer
*   loc: string - location of a picture of the appetizer
*   (this is used to display the picture on the web)
*   pr: real - the price of the appetizer
***********************************************************************/

method body addAppetizer( cat: string, na: string, desc: string, loc: string,
pr: real ) in class Restaurant
{
    o2 MenuCategory curr_category;
    o2 MenuItem new_MenuItem;
    o2 MenuItem curr_MenuItem;

    /* build the menu item we want to add */
    new_MenuItem.name = na;
    new_MenuItem.description = desc;
    new_MenuItem.price = pr;
    new_MenuItem.picture = loc;

    /* see if category already exists */
    for( curr_category in self->Appetizers)
    {
        /* if this is true the category already exists in the appetizers
        set */
        if( curr_category.category == cat )
        {
            self->Appetizers+=set(curr_category);
            curr_category.Items += set(new_MenuItem);
            self->Appetizers+=set(curr_category);

            return;
        }
    }

    /* if no category exists we fall through to here
    * now we must create a new category and add the
    * new menuItem to it
    */
    self->Appetizers+=set(tuple(category: cat, Items:set(new_MenuItem)));
}

/************************************************************
* Restaurant - addMainCourse
* this method adds a main course to the set of main courses
* that are available at the current restaurant.
* if the category is already defined the course is added
* to the existing set or else a new category is created
*  args:
*   cat: string - category of the course
*   na: string - name of the course
*   desc: string - description of the course
*   loc: string - location of a picture of the course
*   (this is used to display the picture on the web)
*   pr: real - the price of the course
***********************************************************************/

method body addMainCourse( cat: string, na: string, desc: string, loc: string,
pr: real ) in class Restaurant
{
    o2 MenuCategory curr_category;
    o2 MenuItem new_MenuItem;

    /* build the menu item we want to add */
    new_MenuItem.name = na;
    new_MenuItem.description = desc;
    new_MenuItem.price = pr;
    new_MenuItem.picture = loc;

    /* see if category already exists */
    for( curr_category in self->Appetizers)
    {
        /* if this is true the category already exists in the appetizers
        set */
        if( curr_category.category == cat )
        {
            self->Appetizers+=set(curr_category);
            curr_category.Items += set(new_MenuItem);
            self->Appetizers+=set(curr_category);

            return;
        }
    }

    /* if no category exists we fall through to here
    * now we must create a new category and add the
    * new menuItem to it
    */
    self->Appetizers+=set(tuple(category: cat, Items:set(new_MenuItem)));
}
MenuItem curr_MenuItem;

/* build the menu item we want to add */
new_MenuItem.name = na;
new_MenuItem.description = desc;
new_MenuItem.price = pr;
new_MenuItem.picture = loc;

/* see if category already exists */
for( curr_category in self->MainCourses)
{
    /* if this is true the category already exists in the set */
    if( curr_category.category == cat )
    {
        self->MainCourses-=set(curr_category);
        curr_category.Items += set(new_MenuItem);
        self->MainCourses+=set(curr_category);
        return;
    }
}

/* if no category exists we fall through to here */
/* now we must create a new category and add the */
/* new menu item to it */
self->MainCourses +=set(tuple(category: cat,Items:set(new_MenuItem)));

/*********************************************************
* Restaurant - addDessert
* 
* this method adds a desert to the set of deserts
* that are available at the current restaurant.
* if the category is already defined the desert is added
* to the existing set or else a new category is created.
* 
* args:
* 
*   cat: string - category of the desert
*   na: string - name of the desert
*   desc: string - description of the desert
*   loc: string - location of a picture of the desert
*       (this is used to display the picture on the web)
*   pr: real - the price of the desert
***********************************************************/
method body addDessert( cat: string, na: string, desc: string, loc: string, pr: real ) in class Restaurant
{
    /* build the menu item we want to add */
    new_MenuItem.name = na;
    new_MenuItem.description = desc;
    new_MenuItem.price = pr;
    new_MenuItem.picture = loc;

    /* see if category already exists */
    for( curr_category in self->Desserts)
    {
        /* if this is true the category already exists in the set */
        if( curr_category.category == cat )
        {
            self->Desserts-=set(curr_category);
            curr_category.Items += set(new_MenuItem);
            self->Desserts+=set(curr_category);
            return;
        }
    }
}
self->Desserts += set(tuple(category: cat, Items: set(new_MenuItem)));

Restaurant - addDrink

* this method adds a drink to the set of drinks
* that are available at the current restaurant.
* to minimize the amount of information we must persist
* we have a global set called DrinkItems. This set contains
* all drinks served at every restaurant. Thus, when we are
* adding a new Drink to the restaurant we first see if it
* exists in this global set and add the reference from this
* set to the set of Drinks in the current object.
* also, for each restaurant we must check to see
* if the category of the drink is already defined. If it is
* the drink is added to the existing set or else a new set
* for the category is created.
* args:
*   cat: string - category of the drink
*   na: string - name of the drink
*   desc: string - description of the drink
*   loc: string - location of a picture of the drink
*     (this is used to display the picture on the web)
*   pr: real - the price of the drink
**********************************************************/
method body addDrink( cat: string, na: string, desc: string, alc: real, pr: real)
in class Restaurant
{
    o2 DrinkItem new_item = new DrinkItem(na, desc, alc);
    o2 Drink addDrink = tuple(price: pr, item: new_item);
    o2 DrinkCategory curr_category;
    o2 DrinkItem curr_item;
    /* determine if drink already exists in global scope */
    for(curr_item in DrinkItems)
    {
        /* if it already exists add this item */
        if( curr_item == addDrink.item)
        {
            for( curr_category in self->Drinks)
            {
                /* if this is true the category already exists in
                * the set */
                if( curr_category.category == cat )
                {
                    self->Drinks += set(curr_category);
                    curr_category.Items += set(tuple(price: pr, item: curr_item));
                    self->Drinks += set(curr_category);
                    return;
                }
            }
        }
    }
    /* if no category exists we fall through to here
    * now we must create a new category and add the
    * new menu Item to it
    */
    self->Drinks += set(tuple(category: cat, Items: set(tuple(price: pr, item: curr_item))));
}
return;
}
}
/* if we reach this point then the Drink Item doesnt exist yet */
DrinkItems += set(addDrink.item);

for( curr_category in self- > Drinks)
{
  /* if this is true the category already exists in the set */
  if( curr_category.category == cat )
  {
    self->Drinks-=set(curr_category);
    curr_category.Items += set(addDrink);
    self->Drinks+=set(curr_category);
    return;
  }
}
/* if no category exists we fall through to here
* now we must create a new category and add the
* new menuItem to it */
self->Drinks +=set(tuple(category: cat,Items:set(addDrink)));

/*********************************************************
* Restaurant - addReview
* this method adds a review about the restaurant to the set
* of reviews that already exist.
* args:
* name: string - name of the reviewer.
* rating: integer - an integer rating from 1-10
* about the overal quality and service of the restaurant.
* comment: string - comments about the restaurant.
***********************************************************/
method body addReview( name: string, rating: integer, comment: string ) in class Restaurant
{
  self->Reviews+=set(tuple(name: name, rating: rating, comment: comment) );
};
/*********************************************************
* Restaurant - removeAppetizer
* this method removes the appetizer from the set of appetizer
* for the Restuarant.
* args:
* cat: string - category of the appetizer
* name: string - name of the appetizer
* return:
* true - if appetizer is found and removed
* false - if the appetizer is not found
***********************************************************/
method body removeAppetizer( cat: string, name: string):boolean in class Restaurant
{
  o2 MenuCategory curr_category;
  o2 MenuItem curr_appetizer;

  for( curr_category in self->Appetizers)
  {
    for( curr_appetizer in curr_category.Items where curr_category.category == cat)
if( curr_appetizer.name == name )
{
    self->Appetizers -= set(curr_category);
    curr_category.Items -= set(curr_appetizer);
    if(count(curr_category.Items) > 0 )
        self->Appetizers += set(curr_category);
    return true;
}
}
}
return false;

/**********************************************************
* Restaurant - removeMainCourse
* this method removes the course from the set of main courses
* for the Restuarant.
* args:
*   cat: string - category of the main course
*   name: string - name of the course
* return:
*   true - if course is found and removed
*   false - if the course is not found
***********************************************************/
method body removeMainCourse ( cat: string, name: string):boolean in class Restaurant
{
    o2 MenuCategory curr_category;
    o2 MenuItem curr_course;
    for( curr_category in self->MainCourses )
    {
        for( curr_course in curr_category.Items where
            curr_category.category == cat )
        {
            if( curr_course.name == name )
            {
                self->MainCourses -= set(curr_category);
                curr_category.Items -= set(curr_course);
                if(count(curr_category.Items) > 0 )
                    self->MainCourses += set(curr_category);
                return true;
            }
        }
    }
    return false;
}

/**********************************************************
* Restaurant - removeDesert
* this method removes the desert from the set of deserts
* for the Restuarant.
* args:
*   cat: string - category of the desert
*   name: string - name of the desert
* return:
*   true - if desert is found and removed
*   false - if the desert is not found
***********************************************************/
method body removeDesert ( cat: string, name: string):boolean in class Restaurant
{
    o2 MenuCategory curr_category;
    o2 MenuItem curr_desert;
for( curr_category in self->Desserts)
{
  for( curr_desert in curr_category.Items where
       curr_category.category == cat)
  {
    if( curr_desert.name == name )
    {
      self->Desserts -= set(curr_category);
      curr_category.Items -= set(curr_desert);
      if(count(curr_category.Items) > 0 )
        self->Desserts += set(curr_category);
      return true;
    }
  }
}
return false;

method body removeDrink ( cat: string, name: string):boolean in class Restaurant
{
  o2 DrinkCategory curr_category;
  o2 Drink curr_drink;

  for( curr_category in self->Drinks)
  {
    for( curr_drink in curr_category.Items where
         curr_category.category == cat)
    {
      if( curr_drink.item->getName == name )
      {
        self->Drinks -= set(curr_category);
        curr_category.Items -= set(curr_drink);
        if( count(curr_category.Items) > 0 )
          self->Drinks += set(curr_category);
        return true;
      }
    }
  }
  return false;
}

method body removeReview ( name: string ):boolean in class Restaurant
{
for( curr_review in self->Reviews)
{
    if( curr_review.name == name )
    {
        self->Reviews -= set(curr_review);
        return true;
    }
}
return false;

method body print in class Restaurant
{
o2 MenuCategory curr_category;
o2 MenuItem curr_item;
o2 DrinkCategory curr_drink_category;
o2 Drink curr_drink_item;
o2 tuple( name: string, rating: integer, comment: string ) curr_review;

self->Establishment@print;
printf("APPETIZERS \n" unt;
for( curr_category in self->Appetizers )
{
    printf("Category: %s \n", curr_category.category);
    for( curr_item in curr_category.Items )
    {
        printf(" Item name: %s, price: $%.2f \n", curr_item.name,
            curr_item.price);
    }
}
printf("MAIN COURSES\n");
printf("----------\n");
for( curr_category in self->MainCourses)
{
    printf("Category: %s \n", curr_category.category);
    for( curr_item in curr_category.Items )
    {
        printf(" Item name: %s, price: $%.2f \n", curr_item.name,
            curr_item.price);
    }
}
printf("DESSERTS\n");
printf("-------\n");
for( curr_category in self->Desserts)
{
    printf("Category: %s \n", curr_category.category);
    for( curr_item in curr_category.Items )
    {
        printf(" Item name: %s, price: $%.2f \n", curr_item.name,
            curr_item.price);
    }
}
printf("DRINKS\n");
printf("------\n");
for( curr_drink_category in self->Drinks)
{
    printf("Category: %s", curr_drink_category.category);
}
for( curr_drink_item in curr_drink_category.Items )
{
    printf("\tItem name: %s, price: $%.2f\n", curr_drink_item.item->getName, curr_drink_item.price);
}
}
printf("REVIEWS\n");
printf("-----\n");
for( curr_review in self->Reviews)
{
    printf("Name: %s, rating: %d, comments: %s\n", curr_review.name, curr_review.rating, curr_review.comment);
}
};

end
Application Flow Diagram:
This diagram shows the flow of the application that is used to interact with the database.
**Functions (Description):**

The functions are primarily used as a way to deal with the management of Persistent Objects. They are used for creating and removing a Restaurant/Bar/CoffeeShop from the Persistent sets of the respective class objects. They are also used to select an object from the persistent set, because once a user has a reference to the object then that user can apply its methods and have the changes be persistent. There are also functions to print out members of each of the persistent sets. There are two reasons that we created functions to do this instead as programs (that are part of a larger application).

1. The documentation was extremely contradictory and said that program could return values (p. 65, “programs can return values”), but when we created a program that returned a value and then attempted to load it into o2, o2 gave the error message that “programs cannot return a value”. But our programs needed to return a value (either a boolean for the remove object functions, or else actual objects for the create and get functions) so they are done as functions.

2. The are functions that will be used by any program accessing using our database and by making them functions (and thus part of the database and not tied to a specific “application”) they can be used by many applications, such as a textual based system and a web front end.

3. They seemed more logically connected to the underlying data schema than to the application using it.

**Function Names:**
(for detailed descriptions see section: Functions(O2C))

```
addRestaurant(n: string, h:string):Restaurant;
addBar(n: string, h:string):Bar;
addCoffeeShop(n: string, h:string):CoffeeShop;
printRestaurants;
printBars;
printCoffeeShops;
getRestaurant(n:string):Restaurant;
getBar(n:string):Bar;
getCoffeeShop(n:string):CoffeeShop;
removeRestaurant(n:string):integer;
removeBar(n:string):integer;
removeCoffeeShop(n:string):integer;
```
Functions (Typical usage):

Creating a new Restaurant:
The user will use the addRestaurant method passing the name and hours of the new restaurant, which will return an instance of class Restaurant that also references the newly created Restaurant in persistent storage. This can then be used to modify/add information to the Restaurant by calling class methods on the object.

Modify an existing Restaurant
Use the function getRestaurant("name of restaurant") to get an instance of class Restaurant that references a Restaurant object in the persistent set Restaurants. Then use the class methods on that object.

Delete an existing Restaurant
Use the function removeRestaurant("name of restaurant") to remove the restaurant of the given name from the persistent set Restaurants. Note: the restaurant must have either 0 or 1 location (ie for a restaurant chain) so that not more than one “physical” restaurant can be deleted”. So if there is more then one location for a restaurant, the other location must first be removed using the class function removeLocation(…).
Functions (O2C):

/*~~~~~~~~~~~~~~~~~~~~~~~~~FUNCTION SPECIFICATIONS~~~~~~~~~~~~~~~~~~~~* /
function addRestaurant(n: string, h: string): Restaurant;
function addBar(n: string, h: string): Bar;
function addCoffeeShop(n: string, h: string): CoffeeShop;
function printRestaurants;
function printBars;
function printCoffeeShops;
function getRestaurant(n: string): Restaurant;
function getBar(n: string): Bar;
function removeBar(n: string): integer;
function removeCoffeeShop(n: string): integer;

/* Note: remove only will remove the bar/restaurant/coffee if it has 0 or 1
location
and removes/checks first one found with that name*/
/**
removes return : 1 if removed, 0 if not found, -1 if too many locations;
*/

/*~~~~~~~~~~~~~~~~~~~FUNCTION METHODS~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~* /

****************************************************************************
* function - removeRestaurant
*
* Removes a restaurant object from the persistent set of restaurants.
*
* args: n: string - name of the restaurant
*
****************************************************************************
function body removeRestaurant(n: string): integer
{
    o2 Restaurant r;
    o2 integer returnval = 0;
    for (r in Restaurants where (r -> getName() == n))
    {
        /* now check the number of locations */
        if (count(r -> getLocations()) > 1)
        {
            return -1;
        }
        else
        {
            Restaurants -= set(r);
            return 1;
        }
    }
    return 0;
}

****************************************************************************
* function - getRestaurant
*
* returns the first occurrence of the restaurant with the same name
*
* args: n: string - name of Restaurant desired to retrieve
*
* return type: Restaurant - returns an object pointing to that restaurant which allows
operations to be performed such as adding appetizers, maincourses, desserts, etc.

**************************************************************************

```javascript
function body getRestaurant(n:string):Restaurant {
  o2 Restaurant r;
  for (r in Restaurants where (r->getName() == n)) {
    return r;
  }
  return nil;
}
```  

**************************************************************************

```javascript
function body getBar(n:string):Bar {
  o2 Bar b;
  for (b in Bars where (b->getName() == n)) {
    return b;
  }
  return nil;
}
```  

**************************************************************************

```javascript
function body getCoffeeShop(n:string):CoffeeShop {
  o2 CoffeeShop cs;
  for (cs in CoffeeShops where (cs->getName() == n)) {
    return cs;
  }
}
```  

**************************************************************************
* function - addRestaurant
*
* Adds the name and hours of a restaurant to the persistent set of restaurants.
* args: n:string - name of the restaurant desired to be added
* h:string - hours of the restaurant desired to be added
* return type: Restaurant - returns the object pointing to that restaurant
**************************************************************************/
function body addRestaurant(n:string, h:string):Restaurant
{
    o2 Restaurant r = new Restaurant(n,h);
    Restaurants += set(r);
    return r;
};

/*******************************************************************************
* function - addBar
*
* Adds the name and hours of a bar to the persistent set of bars.
* args: n:string - name of the bar desired to be added
* h:string - hours of the bar desired to be added
* return type: Bar - returns the object pointing to that bar
*******************************************************************************
function body addBar(n: string, h:string):Bar
{
    o2 Bar b = new Bar(n,h);
    Bars += set(b);
    return b;
};

/*******************************************************************************
* function - addCoffeeShop
*
* Adds the name and hours of a coffeeshop to the persistent set of coffeeshops.
* args: n:string - name of the coffeeshop desired to be added
* h:string - hours of the coffeeshop desired to be added
* return type: CoffeeShop - returns the object pointing to that coffeeshop
*******************************************************************************
function body addCoffeeShop(n: string, h:string):CoffeeShop
{
    o2 CoffeeShop cs = new CoffeeShop(n,h);
    CoffeeShops += set(cs);
    return cs;
};

/*******************************************************************************
* function - printRestaurants
function body printRestaurants
{
    o2 Restaurant r;
    for (r in Restaurants)
    {
        r->print;
    }
}

function body printBars
{
    o2 Bar b;
    for (b inBars)
    {
        b->print;
    }
}

function body printCoffeeShops
{
    o2 CoffeeShop cs;
    for (cs in CoffeeShops)
    {
        cs->print;
    }
}