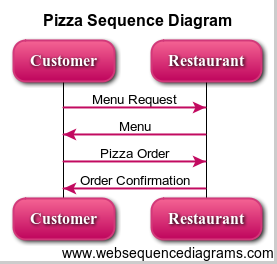
The Pizza Protocol

# CSCI 4760/6760

**What is this?** Suppose we want to order a pizza remotely, either for pickup at the restaurant or delivery. The sequence diagram for either of these scenarios is shown at right. (Thanks to websequencediagrams.com for their free sequence diagram drawing service.)

In this process the Menu Request does not contain any extra information besides the request for a menu, but the Pizza Order must contain information such as “two slices with Sicilian crust and pepperoni and mushroom toppings.” We will define precise contents of each of the four message types as an in-class exercise.

Two key properties of this process are worth noting:

1. This is a classic ***client-server*** interaction. The server (Restaurant) must first be listening for an incoming request . The client (Customer) initiates the actual interaction.
2. The process is ***stateless***; that is, neither the restaurant nor the customer needs to remember any information between two request-response interactions. (The restaurant needs to temporarily remember the data contents of a Pizza Order message while computing the response, but not between two requests). Indeed, a customer who already knows what they want can place an order without first requesting a menu. Each of these two request-response pairs is called a ***transaction***.
3. The menu request operation is ***idempotent***; that is, submitting multiple menu requests has the same effect as a single menu request. However, ordering pizza is not idempotent!

# Network Protocols to Use:

The data contained in menus and pizza orders is *application-layer* data. If we were to develop a Web application for ordering pizzas, the Web server and browser would send this data in HTTP request and response messages. On the other hand, if we chose to build a dedicated app, we would likely create a specific protocol for this application. In this assignment we will implement an application-specific protocol: the Pizza Protocol.

According to the Protocol Layering Principle, a protocol at one layer calls on the services of a protocol at the next layer down. In order for the Pizza Protocol to pass messages between the client and the server, it needs to call on the services of a transport protocol. For this first assignment, we will use the TCP protocol, running on port 2014. You can learn about TCP socket programming in chapter 2 of your textbook.

# Steps for completing this assignment:

1. In class, we will define Java classes to represent the four message types shown in the sequence diagram above. Also, we will discuss how to serialize each of these messages into a byte array for transmission across the network, and deserialize the bytes back into objects at the receiving end.
2. You will next through the following exercises, available on eLC:

* Extreme Programming
* Java Socket Programing
* Currency Formatting
* Java Inheritances

Because these are exercises, you are welcome to work on them with a partner.

1. Working as an individual, you will develop the message Java classes using Test-Driven Development. I will provide some JUnit test code. These must reside on your VM in a folder named Assn1.
2. Working as an individual, you will develop the client and server application programs. In class we will develop a standard test menu. I will provide a test server against which you can test your classes. The client application must have a friendly user interface. For extra credit for undergrads (required for CSCI 6760 students), create a graphical user interface using Swing or JavaFX.
3. To get full credit, you must document successful client-server interactions with at least two other students running your server, and two other students running your client. These can be the same two other students.
4. You do not need to submit this assignment on eLC; we will grade it on your VM.

## Grading Rubric

|  |  |
| --- | --- |
| Ant build.xml file that builds and runs JUnit tests | 30 |
| Message classes pass JUnit tests | 30 |
| Student PizzaClient class interacts with own or Dr. Dan’s PizzaServer | 20 |
| Student PizzaClient class interacts with two other student PizzaServers | 10 |
| Student PizzaServer class interacts with two other student PizzaClients | 10 |