

**Instructor:** Mohsen D. Ghassemi

**Team:** Group 2

**Members:** Yidi Xu, Shuojia Shen, Wei Xu

**Project:** Online Client/Server Android Restaurant Application

**Date:** April 18th, 2018

* **Introduction**

With the rapid development of information technology, Android application have been increasing in recent years.

The advantage of the Android application:

- Mobile application is convenient to carry

- Global partnerships and large install base

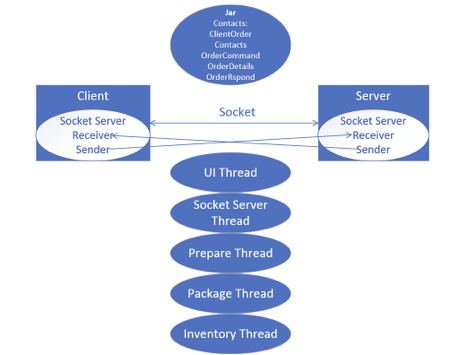
- Powerful development framework

- Open marketplace for distributing apps

Based on these advantages, we have developed an Online Client/Server Android Restaurant Food Ordering Application. Assumption has been made that the restaurant is using internet technology to run its business. The customer could use mobile devices ordering food online and pick up when the food is ready.

* **Technology Overview**

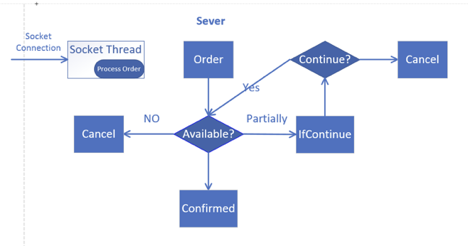
In general, we used Producer-Consumer pattern, socket communication, multiple threads, and Handler & Looper to develop this application. The main activity shows as follow:



We included one Jar package in both Client and Server applications, which includes: ClientOrder, Contacts, OrderCommand, OrderDetails, OrderRespond. In addition, we built Socket to communicate with these two applications. In both ends, we created one socket receiver thread to receive from the sender from the other side. For example, we could receive OrderCommand from Client Socket Sender Thread in Server side, and we could receive OrderRespond from Server Socket Sender in Client side. Furthermore, we created five threads: UI Thread, which is to update UI; Socket Server Thread, which is to communicate with both applications; Prepare Thread, which is to prepare orders from customers; Package Thread, which is to package and deliver orders to customers; Inventory Thread, which is to check availability of orders.

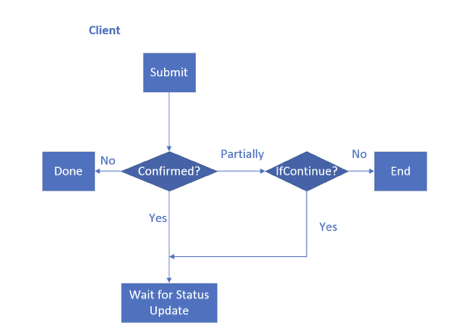
* **Flow Charts**

1. Server



The server got connected with client by socket server thread. Order should be check availability, if it is available, the order will be confirmed, and send OrderRepond: Confirmed to client. If the order is fully unavailable, then it will be canceled automatically. If the order is partially available, it will send OrderRepond: IfContinue to client. The client will decide whether continued. If yes, then the order will be check availability again, and send OrderRepond: Confirmed to client. If no, the order will be canceled.

1. Client

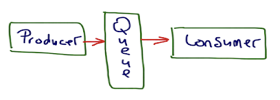


The client submitted the order, if it is confirmed, then he/she will wait for order status update. If the order is not confirmed, it will be canceled automatically. If the order is partially available, the client should choose whether continue. If yes, the client will wait for order status update. If not, the order will be canceled.

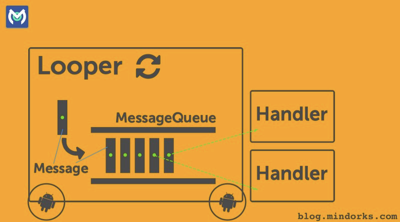
* **Design Pattern**

1. Producer-Consumer Pattern

The Producer Consumer pattern is an ideal way of separating work that needs to be done from the execution of that work. the Producer Consumer pattern contains two major components, which are usually linked by a queue. This means that the separation of the work that needs doing from the execution of that work is achieved by the Producer placing items of work on the queue for later processing instead of dealing with them the moment they are identified. The Consumer is then free to remove the work item from the queue for processing at any time in the future. This decoupling means that Producers don't care how each item of work will be processed, how many consumers will be processing it or how many other producers there are. It's a fire and forget world as far as they're concerned. Likewise, consumers don't need to know where the work item came from, who put it in the queue, and how many other producers and consumers there are. All they need to do is to grab some work from the queue and process it. We used LinkedBlockingQueue to achieve the goal.



1. Handler and Looper

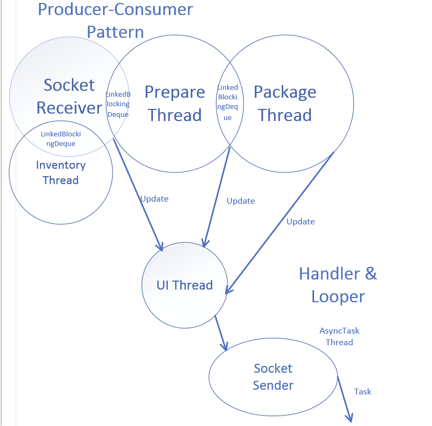


Reference: <https://blog.mindorks.com/android-core-looper-handler-and-handlerthread-bd54d69fe91a>

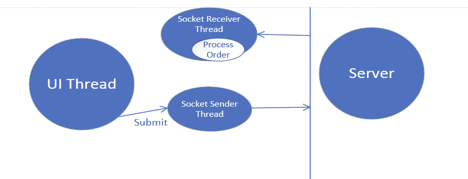
* MessageQueue is a queue that has tasks called messages which should be processed.
* Handler enqueuers task in the MessageQueue using Looper and also executes them when the task comes out of the MessageQueue.
* Looper is a worker that keeps a thread alive, loops through MessageQueue and sends messages to the corresponding handler to process.

1. Multiple Threads

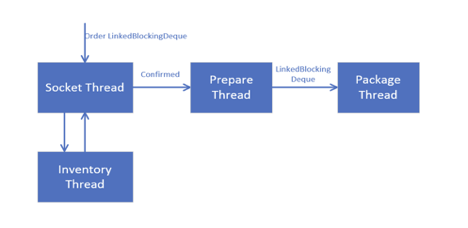
* Main

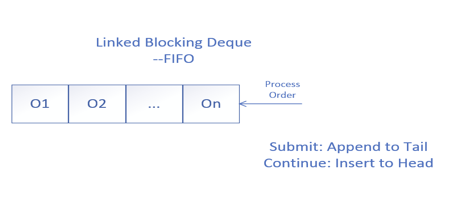


* UI Thread



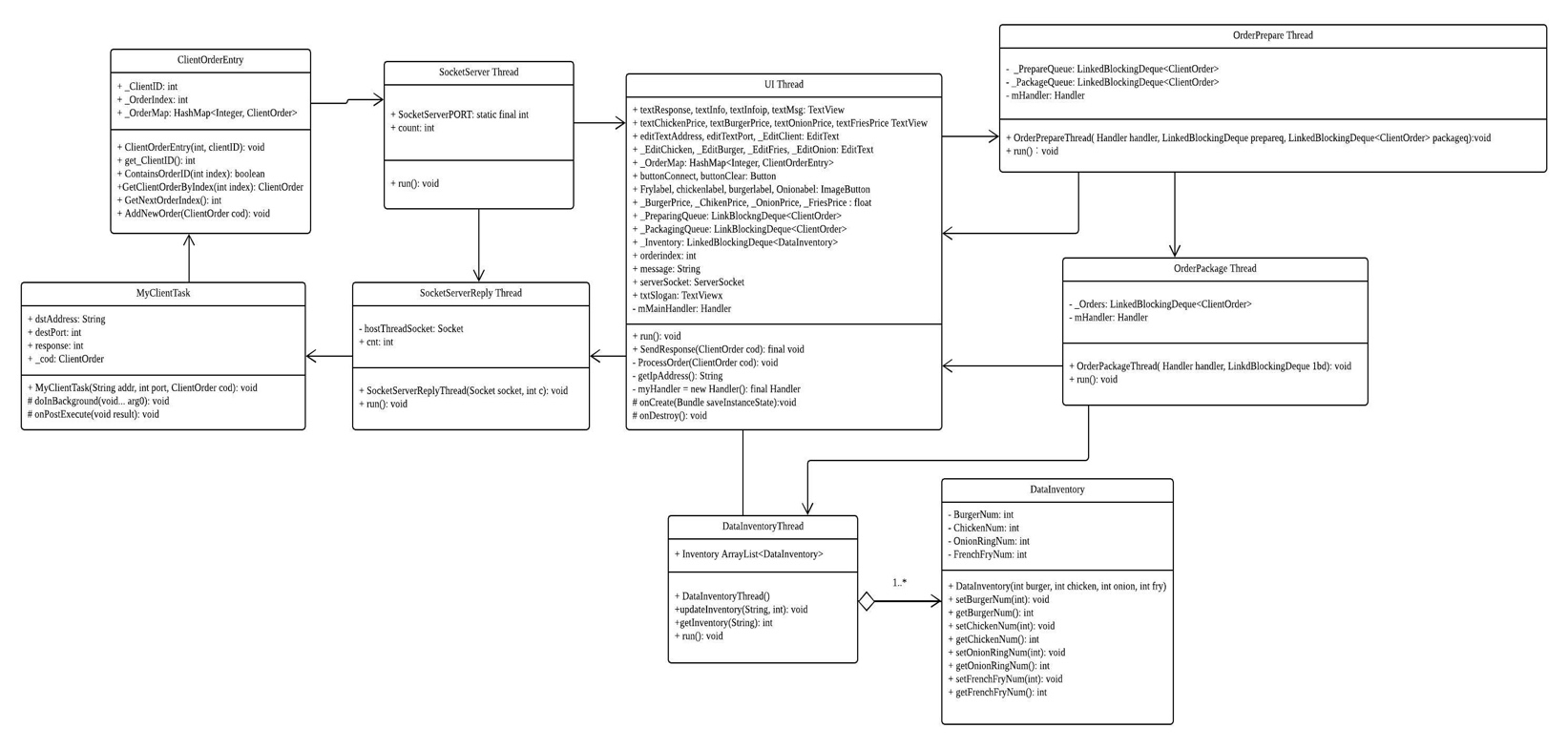
* Linked Blocking Queue





* **Unified Modeling Diagrams**

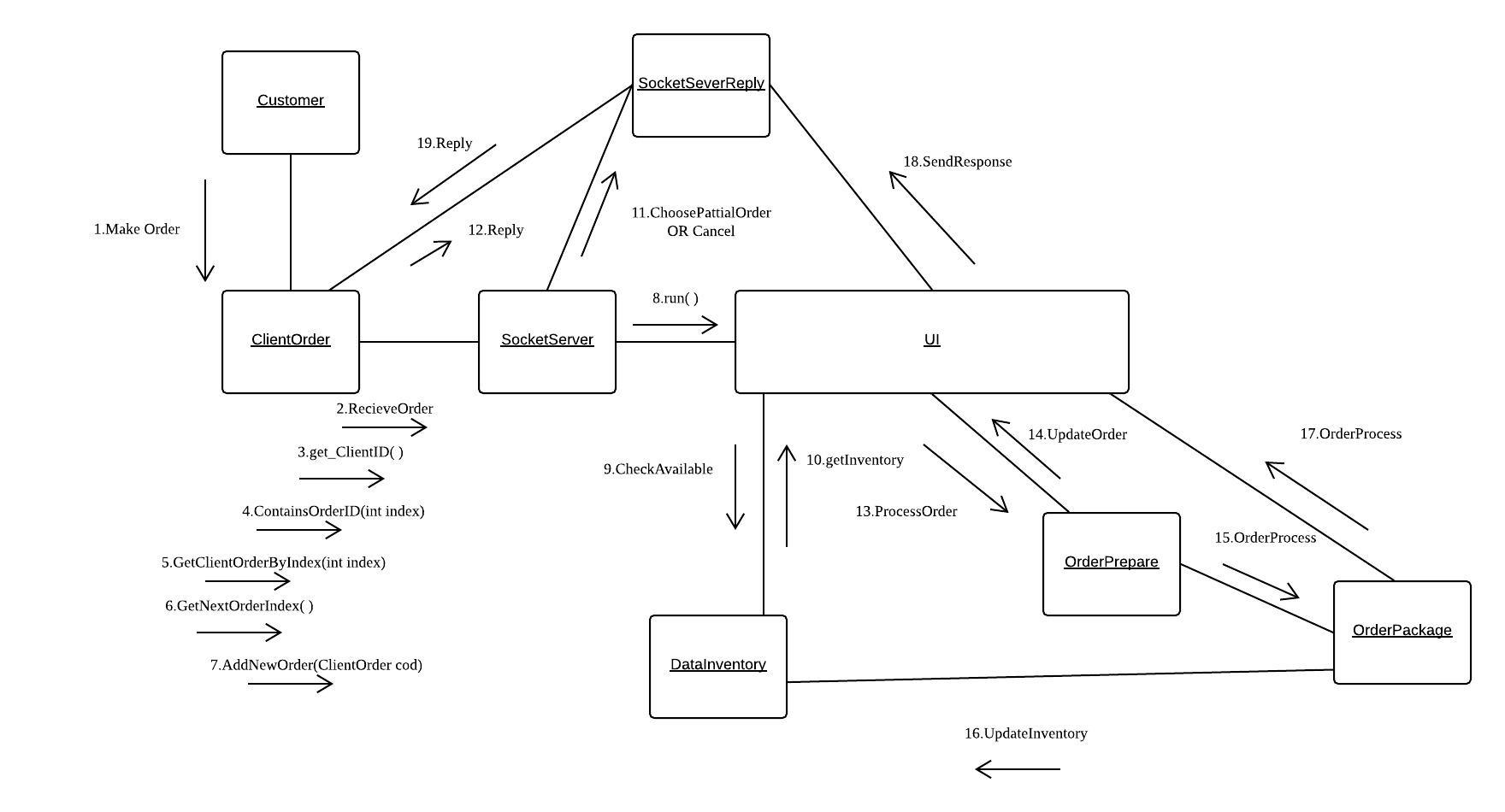
1. Class Diagram



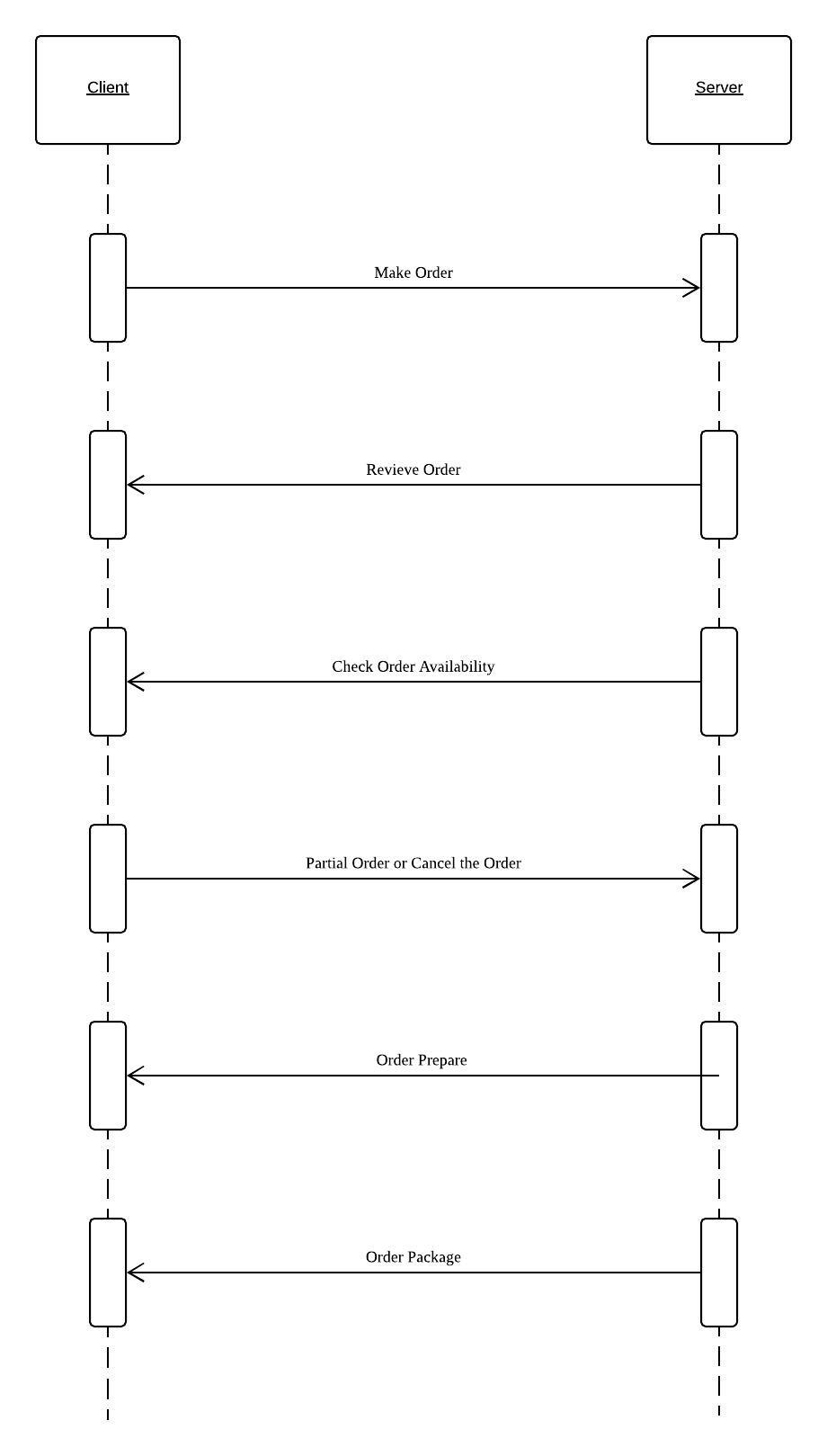
1. Use Case



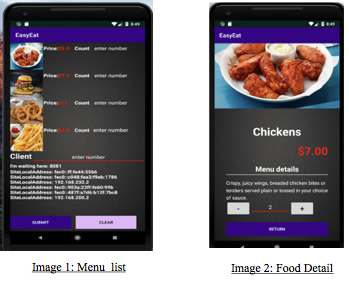
1. Collaboration



1. Sequence

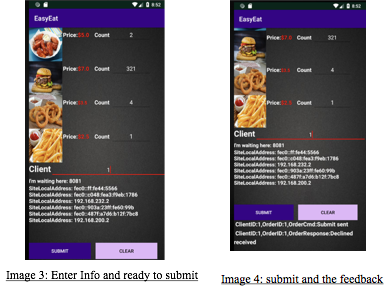
****

* **Test Units**
  1. Menu List (enter the number of the item here directly)
  2. Food Detail （ Click on the picture and jump into the food detail）

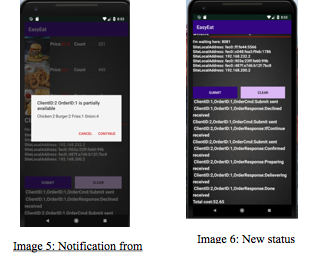
****

* 1. Enter the Client\_ID and prepare to submit (Image 3)
  2. Click on submit button to place your order and see the the feedback from server(Image 4)

Because of our setting, the first order will be declined.



* 1. Change your client id and submit again. Now you can see the notification from server part. (Image 5)
  2. If you choose continue, the new status will be updated. (Image 6)



* 1. In the server end, the status will update as well. (Image 7 & 8)

