# Coding Style Guidelines

The team will utilize a predetermined style format determined by checkstyle. Along with distinct android protocol

## Checkstyle enforced style (major requirements)

All Classes and members are properly documented with JavaDoc.

a. Description for every parameter
b. Description for return type
c. Includes at a minimum a brief description of what it is or does abstractly
d. Author's Name or uw net id for reference

1. Max File length- 2000 lines
2. Max method length- 150 lines
3. Max parameters- 7 lines
4. Max line length- 100 lines
5. Final constants are in ALL_CAPS
6. Local variables, parameters, types, method names in camelCase
7. There must be a white spaces after "," ";" and type-casts
8. No whitespace before "." ";" "++" "--" ")"
9. No whitespace after "("
10. Follow practical boolean zen (i.e. disallows if(foo == true), (foo || true), !false)
11. Utility Class should not have a public constructor

## The following are not enforced by checkstyle

1. Each package in both project must relate to a particular module of the application
    a. IE: login
2. All Activity class names will end in Activity
3. All Fragment class names will end in Fragment
4. Fragments will not do any heavy processing or network calls.
5. Fragments must be initialized with arguments by putting a Bundle in setArguments.
6. If an Activity requires the use of user specific data then then the Activity must extend DineOn(User|Restaurant)Activity.
7. The DineOnUserActivity is used as parent activity that manages overall state of the User (Customer) during application usage.
    a. All general usage that reflects across all user involved classes shall be placed in this Activity
8. The DineOnRestaurantActivity is used as a parent activity that manages overall state of the of the Restaurant user.

a. All general restaurant activity usage shall be placed in this parent class

## Communication Design Pattern

- Our current communication design pattern has been oriented to work with Android's naturally stateless interActivity communication and Parse cloud utilization. Instead there will be distinctive state objects that are passed around between activities and updated independently. This technique relies on the centralization of our parse cloud object. Once an activity is ready to start another activity via an intent, it stores our state objects within the intent it just created. However Android restricts this storage operation to behave more like a deep copy. To mitigate mis-synchronization, we will ask the network for a fresh copy of the same state object. To capture real time updates we will use dynamic broadcast receivers that are started and stopped on Activity start and stop. Later phases we will implement some form of Android persistent storage as a state saving process. Another potential option is Parse providing caching, but its capabilities have not been assessed at this time.

## Code Reviews

The team did not figure out how to do Github code reviews until later on the phase. There was one significant push to the repository. Here are some key review notes from that initial commit.

- There was white space inconsistency in different class
- There was too much variation in methods to listening for user input events
- data types were filled in with space filling type String. It was not clear where to replace with the real data types
- A lot of the code was not very well documented. It was more of a mass skeleton code push for the rest of the developers.
- Some classe were missing obvious methods
- Structure of the classes should be consistent for ease of use
- createDineOnUser
    - Async concern
        - Will Activity be alive at the time when the Async call returns?
        - If not we need to deal with invoking the callback on an Activity that no longer exists
    - Handle exception when it is not null in signupcallback

- Update all UI classes to support QR code.
- Handle all exception cases onResult and onReceive in UI for pushe and QRcode stuff.
- Write Request class for user to send restaurant requests for bill,water, etc.
- Write/finish Javadoc for all classes and also for methods that are not overriden

All other reviews were done via Pull Request on GitHub:

https://github.com/blasv/DineOn/pulls?direction=desc&page=1&sort=created&state=closed