# GRASP Patterns - Baylor Burgers

Listed below our the concepts in our design class diagram:

## Menu

This class has not been seriously implemented in our prototype, but we can expect this class to be responsible for keeping track of the different Food Description objects available at a certain location/time/etc by using category as an indirection between the two to lower coupling. Thus this class would be considered an Information Expert.

## Category

This class has not been implemented yet but we can expect it to be an information expert. Also making these categories allows for a more understandable code design, helping with high cohesion and low coupling.

## Food Description

This class is responsible for tracking the attributes of a particular menu item, thus making this an Information Expert. Also, we chose to separate the Food Description from the Cart Items so that we would not have to track multiple copies of the same data from the Food Description class. This would be an example of high cohesion as we are separating data into multiple classes to reduce redundancy.

## Cart Item

This class is responsible for keeping track of how many items of a particular Food Description a customer has ordered as well as tracking the special requests placed by customers; therefore, this is an example of an Information expert. Also, we chose to separate the Food Description from the Cart Items so that we would not have to track multiple copies of the same data from the Food Description class. This would be an example of high cohesion as we are separating data into multiple classes to reduce redundancy.

## Cart

This class is responsible for keeping track of the items in a customer's order while they have not finished adding new items or are still ordering. This is an example of an information expert as it keeps these items accounted for.

## Manager

This class is responsible for the basic login system and is mostly a class so that our GUI classes don't have to track it. This is an example of pure fabrication as we did not ***need*** this class but we decided it would be easier to understand if it were separate.

## Payment

This Payment class is responsible for handling all payment events, however, we decided to separate the card payment out into a subclass of this general type, through polymorphism, to make sure our design is cohesive and supports low coupling.

Card

This class is responsible for handling Payment events made with Card, we chose to separate this from the general type of Payment because it will work fundamentally differently and thus demonstrates higher cohesion.

LoginUI and other unimplemented UI classes

These classes are responsible for handling and managing user interaction with the system, and thus can be identified as Controller classes, these classes are how the user will interact with the menu and other objects/classes within our system. This also can be described with the ‘Don’t Talk to Strangers’ GRASP principle as we don’t want the user handling the internals of our system. As we further implement these classes, this may end up being where most of the internal instances of classes are created, which would make some of the classes Creator classes; however, we have yet to implement some of these functionalities thus far and that remains to be seen.