UML Diagrams: MattSt
Jen Yu, Brooke Jin, Bryan Chan

[for interest of space & time, only modifications of the classes along with new classes are shown. for older methods, refer to the older UML diagrams.]

PekingWong

Customer d;
Waiter ling
Restaurant pekingWong
Kitchen k

---

setup(): sets up the size of the world & instantiates variables.
draw(): calls the display() methods of the variables
void mouseClicked(): gets the Waiter moving and updates his/her status
void mousePressed(): utilized for dragging the customer in the world.
void mouseDragged(): Another part of the dragging facility, changes state of Customer d during drag
void mouseReleased(): checks if any of the tables have been clicked and then updates variables & activates Waiter functionality.

Table

Customer c
Order order
int x, y, state, tableNum

Table(int num, int setX, int setY): Set the tableNum & the (x,y) position
void display(): Displays table shape.
boolean overTable(): checks if the mouse is over the table
boolean inside(float currX, float currY) checks if coords are inside table.
void setTable(Table t), setCust(C in); setOrder(Order o)
Order getOrder(), Customer getCust()

Draggable

float bx, by
float xOffset, yOffset
int size
boolean overBox, locked

Draggable(int num): sets the size to the parameter. All of the booleans default to false. bx and by are equal to the width & height of the world divided by 2.
void display(): sets overBox based on if the mouse cursor is over the Draggable.

Customer

extends Draggable

private String name
private Table table
private int mood, VIPNum, state
private boolean served
int origX, origY

Void display(): changes the position based on state.
Void checkState(): If locked then modify x & y coors (drag)
Customer(): calls superclass constructor with param 20 & sets bx & by to default coors.
void setTable(Table t): sets the table occupied by customer
void setState(int i): sets state
Void nowServed(): served is #t
Table getTable(); int getMood()
Deletions: overloaded constructor, toString()

Waiter

private ArrayList<Customer> customers
private Customer currCust
private Kitchen k
private ArrayList<Table> tables
private ArrayList<Order> orders
private Order[] finishedOrders
float x, y, xMouse, yMouse
boolean waiterMoves
int state

Void display(): using a for-each loop, displays all tables & customers currently being served
Void update(): checks if mouse clicked on a table, the kitchen, or an order & acts accordingly
Void detAct(): depending on state of table (parameter, not shown sorry) executes an action.
Deletions: assignTable(), findCust(), chooseAction(), getNextOrder()
* Waiter constructor takes in Kitchen

Time

long start, end, elapsed, target

Time(long goalTime): sets target
void startTime(): starts the time by setting start to curr. nanoTime
Void endTime(): ends the time & sets end to the curr. nanoTime
long getElapsed(): ends the time and then returns the difference of end & start in seconds.
boolean atGoal(): checks if elapsed time has surpassed target time
long toSeconds(long time): to sec!

Kitchen

... Deque & Queue ...
Order currOrder
int x, y

Void display(): displays the kitchen and the current finished order
boolean overKitchen(): checks if mouse is over the kitchen
* Waiter has-a Kitchen

Order

protected String dishName
protected int tableNum
Time t
int x, y, state

Void display(): displays an Order with a number representing its table#
boolean overOrder(): Sees if the mouse is over the Order object.

Console

Waiter w
PFont type
PImage paper

Console(Waiter w): Sets up a console.
Void display(): Shows the current stats of all the tables and the waiter's platter.

*Restaurant class essentially just takes care of the wait list

Time (cont'd)

boolean pause
long pauseTime, pauseTimeStart
void endPause(): ends pause
boolean endInterval(): sees if the pause is over 5 sec