January 7, 2014

ICS Final – Design Document

**File Format**

**Ingredients.txt**

<number of ingredients in list>

<ingredient name>

<unit of measurement>

<amount of ingredient>

**Dishes.txt**

<number of dishes in menu>

<name of dish>

<type of dish>

<number of times made>

<cost for dish>

<first ingredient name>

<second ingredient name>

.

.

<name of dish>

.

.

**Suppliers.txt**

<number of suppliers>

<name of supplier>

<number of ingredients to sell>

<ingredient name>

<ingredient cost>

<ingredient name.

<ingredient cost>

.

.

<name of supplier>

.

.

**Class List**

**Assume Accessors/Mutators for all fields in all classes**

**Ingredients**

-responsible for the characteristics of an ingredient

Fields:

* name- String
* number – int
* unitMeasurement – String

Methods:

* Constructor – initialize all fields

**Meat –** subclass of Ingredient

**Dairy** – subclass of Ingredient

**Vegetable** – subclass of Ingredient

**Fruit** – subclass of Ingredient

**Grain** – subclass of Ingredient

**Seafood** – subclass of Ingredient

**Spice –** subclass of Ingredient

**Drawer**

Fields:

* storedIngredient – Ingredient
* amountLeft - double

Methods:

* Constructor – initialize all fields
* addAmount – takes in the amount to add to storage, and updates the total. Returns a Boolean to indicate whether top off was successful.
* deductAmount – takes in the amount to deduct from storage, and updates the total. Returns a Boolean to indicate whether deduction was successful or not.

**Inventory**

Fields:

* ingredientList – Drawer[]

Methods:

* addIngredient – adds amount of ingredient to specified drawer
* checkInventory – checks to see if all ingredients specified are present

**Dish**

Fields:

* Name - String
* menuNumber - int
* ingredients [ ] - Ingredients
* cost - double

Methods:

* Constructor – initialize all fields
* checkInventory (ingredients [ ]) – returns a Boolean saying whether or not the list of ingrdients necessary are all available
* CompareToCost(dish) – returns a double to compare the cost between two dishes.
* toString- Returns a String specifying the dish’s name and cost, followed by the ingredients needed

**Drink** – subclass of Dish

Fields:

* alcoholic - boolean

Methods:

* Constructor – initializes all fields
* CompareToCost(drink) – returns a double to compare the cost between two drinks.
* toString- returns the values of all fields in this object in an organized format.

**Entrée** – subclass of Dish

Methods:

* CompareToCost(Entree) – returns a double to compare the cost between two entrees.

**Appetizer** – subclass of Dish

Methods:

* CompareToCost(Appetizer) – returns a double to compare the cost between two appetizers.

**Dessert** – subclass of Dish

Methods:

* CompareToCost(desert) – returns a double to compare the cost between two deserts.

**Produce**

-responsible for storing the name and unit price of an ingredient

Fields:

* ingred - Ingredient
* unitCost - double

Methods:

* toString – prints out the ingredient and its cost

**Supplier**

-responsible for storing the name of producer, and a list of Produce objects

Fields:

* producerName - String
* produceList – produce[]

Methods:

* toString – prints out the name, followed by a list of products and their price
* findSuppliersByItem – takes in the desired item to purchase, and returns a string with a list of all producers that sell that product with the price
* findSupplierByName – takes in the name of a supplier, and returns a string with the list of supplies they have
* buyFromSupplier – takes in the supplier name and amount of product they want and returns the cost

**Budget**

Fields:

* balance - double

Methods:

* addRevenue – access the cost of a dish once it is ordered and add it to the account balance
* removeRevenue – access the cost of the ingredient purchased from supplier and deduct the money from the account balance
* toString – return the balance statement from the account

**RestaurantDatabase**

Fields:

* dishList – Dish[]
* recentlyMade – Dish[]
* storage – Inventory
* supplierList – Supplier[]
* budget – Budget

Methods:

* loadInventory – load inventory from text file
* loadDishes – load dishes from text file
* loadSuppliers – load suppliers from text file
* loadBudget – load budget from text file
* saveInventory – save inventory to text file
* saveDishes – save dishes to text file
* saveSuppleirs – save suppliers to text file
* saveBudget – save budget to text file
* buyIngredient – takes in a supplier name, ingredient name and amount and buys the ingredient if there is enough money
* cookDish – takes in a dish name and cooks that dish if all ingredients are present, and adds the cost of the dish to the budget
* searchIngredientByName – takes in a name and returns the index of the drawer object with that name
* searchDishByName – takes in the name and returns the index of the dish object with that name
* findSupplierByName – takes in supplier name and returns the index of the supplier object with that name
* sortIngredientsAlpha – sorts ingredients in alphabetical order
* sortDishesAlpha – sorts dishes in alphabetical order
* printMostRecent – prints the most recent dishes
* printAllIngredients – prints out all ingredients to the screen
* printAllDishes – prints all dishes out to the screen
* printSpecificDish – prints a specific dish specified by the user
* printAllSuppliers – prints out all suppliers to the screen
* printBudget – print budget info to the screen

**RestaurantRunner**

Methods:

* Main – display menu and perform actions according to the user’s choices