RestaurantV1

Roel Janssen

March 3, 2017

1 Introduction

This rapport will discuss the necessary changes to the original project RestaurantV1 to the multithreaded project for assignment 1 for course Java3. The theory will not be discussed in detail, but rather the changes made and why those changes were made.

2 Progress and Errors

As enthusiastic students always do, a start was made without a proper design. This section will discuss what problems occurred and how these were solved.

The first decision made was a good one, the Oracle documentation was studied and a theoretical basis was made. After that, the original project was studied. This was still not bad, but a small overview should have been made. This would have helped identifying the necessary changes to be made. Instead, the project was treated as a playground. Instead of 'What needs to be done' the project was treated as 'What would happen if' and 'how could this be done'. This wasn't bad, a lot of knowledge was gained, but no real progress was made. A lot of commented out code was scattered through the project which made the entirety confusing. A clean start was made, and this time with a plan. First the exception for an unknown meal number was changed into a printline, so the application wouldn't crash. After that, the plan was to make three anonymous innerclasses implementing the Runnable class, being a cook, a server and a waiter. This was a rather easy solution, since the work those classes should do was neatly gathered in one or two methods per class.

1. The waiter was supposed to submit the orders by the submitOrder(String ...ordered) method;
2. the cook was to process the orders with the procesOrders() method and prepare the 'meals' using the prepareMeal(int orderNR, int mealNR);
3. The server was to deliver the orders with the getNextMeal() method.

This lead to the next problem: the methods were using parameters, how would these be acquired? This was solved by creating getters and setters in the Restaurant class. This required the subclasses to know the Restaurant, so a constructed requiring a Restaurant class had to be created.
This was on the right track, but the Restaurant class was getting large and confusing. This was solved by moving the subclasses to separate classes. This was a huge improvement for clarity and understandability. Now the methods being for each class had to be put in the run() method, since that is how the thread will be executed. Here the next problem was encountered. Since the run() method has no parameters, how could the necessary parameters be acquired? This meant creating getters and setters again.
The next major problem was an illegal thread state exception. After some research the cause was found. This design tried to instantiate a thread in the Restaurant constructor, and using methods called by the main class to execute the run method of the thread. This meant that a single instance of a thread was executed multiple times. This was impossible, since a thread can only run once. This was fixed by declaring and instantiating the threads in the responsible methods. This way, each time a method was called a new instance was created.

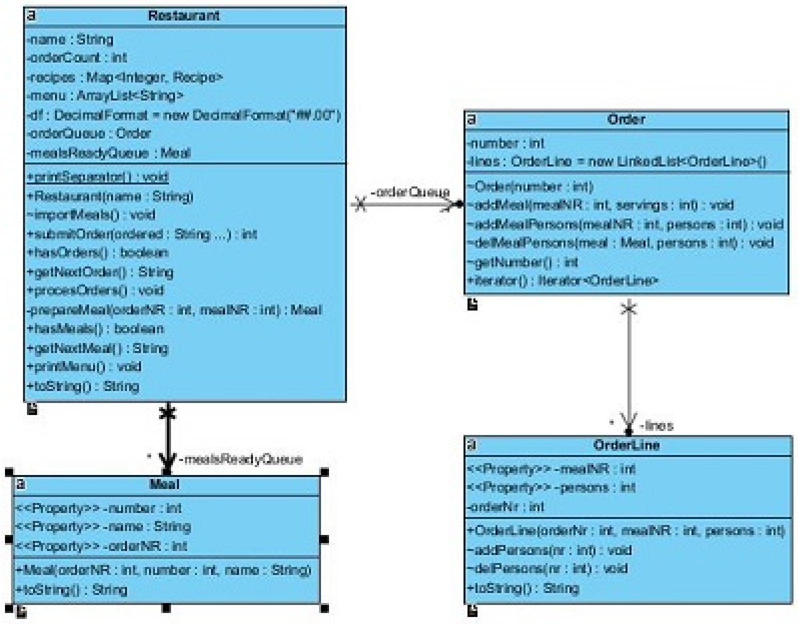
Restaurant
-name : String
-orderCount : int
-recipes : Map<Integer, Recipe>
-menu : ArrayList<String>
-df : DecimalFormat = new DecimalFormat("##.00")
-orderQueue : Order
-mealsReadyQueue : Meal
+printSeparator() : void
+Restaurant(name : String)
~importMeals() : void
+submitOrder(ordered : String ...) : int
+hasOrders() : boolean
+getNextOrder() : String
+procesOrders() : void
-prepareMeal(orderNR : int, mealNR : int) : Meal
+hasMeals() : boolean
+getNextMeal() : String
+printMenu() : void
+toString() : String
-orderQueue
Order
-number : int
-lines : OrderLine = new LinkedList<OrderLine>()
~Order(number : int)
~addMeal(mealNR : int, servings : int) : void
~addMealPersons(mealNR : int, persons : int) : void
~delMealPersons(meal : Meal, persons : int) : void
~getNumber() : int
+iterator() : Iterator<OrderLine>
*
-lines
OrderLine
<<Property>> -mealNR : int
<<Property>> -persons : int
-orderNr : int
+OrderLine(orderNr : int, mealNR : int, persons : int)
~addPersons(nr : int) : void
~delPersons(nr : int) : void
+toString() : String
*
-mealsReadyQueue
Meal
<<Property>> -number : int
<<Property>> -name : String
<<Property>> -orderNR : int
+Meal(orderNR : int, number : int, name : String)
+toString() : String

Figure 1: The original class diagram (reduced to restaurant class)

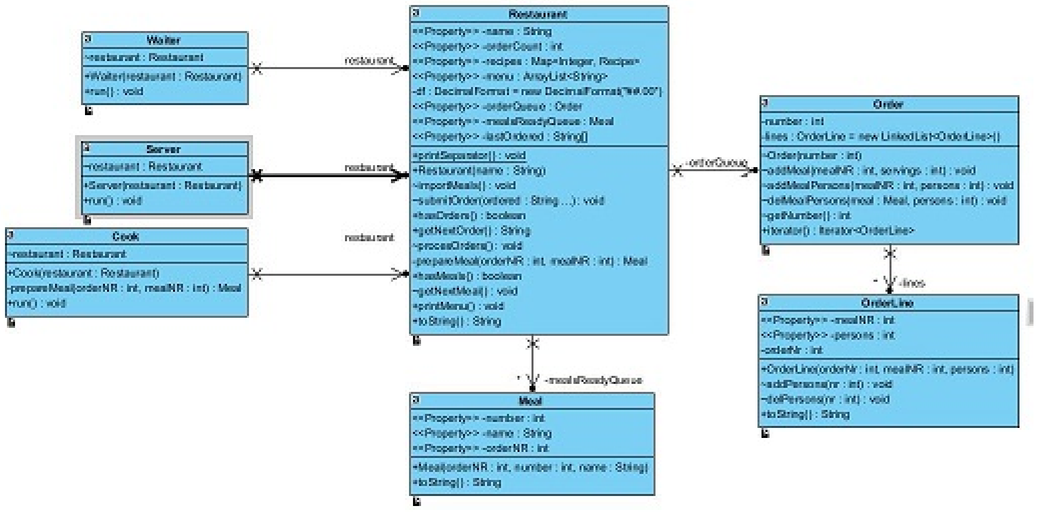
Waiter
~restaurant : Restaurant
+Waiter(restaurant : Restaurant)
+run() : void
restaurant
Server
~restaurant : Restaurant
+Server(restaurant : Restaurant)
+run() : void
restaurant
Cook
~restaurant : Restaurant
+Cook(restaurant : Restaurant)
-prepareMeal(orderNR : int, mealNR : int) : Meal
+run() : void
restaurant
Restaurant
<<Property>> -name : String
<<Property>> -orderCount : int
<<Property>> -recipes : Map<Integer, Recipe>
<<Property>> -menu : ArrayList<String>
-df : DecimalFormat = new DecimalFormat("##.00")
<<Property>> -orderQueue : Order
<<Property>> -mealsReadyQueue : Meal
<<Property>> -lastOrdered : String[]
+printSeparator() : void
+Restaurant(name : String)
~importMeals() : void
~submitOrder(ordered : String ...): void
+hasOrders() : boolean
+getNextOrder() : String
~procesOrders(): void
-prepareMeal(orderNR : int, mealNR : int) : Meal
+hasMeals() : boolean
~getNextMeal() : void
+printMenu() : void
+toString() : String
-orderQueue
Order
-number : int
-lines : OrderLine = new LinkedList<OrderLine>()
~Order(number : int)
~addMeal(mealNR : int, servings : int) : void
~addMealPersons(mealNR : int, persons : int) : void
~delMealPersons(meal : Meal, persons : int) : void
~getNumber() : int
+iterator() : Iterator<OrderLine>
*
-lines
OrderLine
<<Property>> -mealNR : int
<<Property>> -persons : int
-orderNr : int
+OrderLine(orderNr : int, mealNR : int, persons : int)
~addPersons(nr : int) : void
~delPersons(nr : int) : void
+toString() : String
*
-mealsReadyQueue
Meal
<<Property>> -number : int
<<Property>> -name : String
<<Property>> -orderNR : int
+Meal(orderNR : int, number : int, name : String)
+toString() : String

Figure 2: The class diagram after adding thread classes

With these modifications the application was running multithreaded. The only problem was that the ordernumber wasn't updated. This was caused by the lack of a return in the void run() method for the Waiter class. This was fixed by using a setter at the end of the run method to update it. The only thing done at the end was deleting commented code for a neater file.