Pelin Sena VAROL

Yağmur Doğan / 10963330588

Pelin Sena Varol / 64360013520

Şevval Türkyılmaz / 49927389416

**DATA TERM PROJECT**

**GROUP 6**

Which data required to be taken from the user and which data should be constant?

We ask the user what product they want to order and we print the product list. Then, they can enter the id of the product they want and they can see the outputs of MRP tables.

We took Amount On Hand, Scheduled Receipt, Arrival on Week, Lead Time and Lot Sizing Rule values as a constant. We took these values from items.txt file that we created.

Which data structures are utilized? Why?

In the pseudocode, we worked with arrays. We made our algorithm based on arrays. When we were writing our pseudocode, we thought the best choice for the problem is using arrays because we have fixed 10 weeks for theordering process. We used arrays because they are fast for this problem and it’s easy to work with arrays. With using for loop and our algorithm, we find the required MRP outputs for our items.

We used ArrayList data structure in reading data from text file part. When we first get data from the text, we store these datas in ArrayList with type String. Then we used 2DArrays to use the datas for each item. We used ArrayList because the text has fixed-size, so it’s more efficient to use ArrayList in here.

Also, we used object-oriented principles for our project and we handled the possible exceptions while reading data[FileNotFound Exception and IO Exception]

And in the algorithm we handled exceptions with try-catch blocks.

And also there are some places in the algorithm which cause IndexOutOfBoundsException. We handled these exceptions with the control statements. For example we used if(i > leadTime) for the case of [i – leadTime] because it can gives us a negative value. And in the some places of code we used (i + 1 < 10) control statement because i+1 can cause IndexOutOfBoundsException. In our algorithm our maximum i value is 9. So, when it becomes 10, it can cause a problem. Also we used some similar control statements to control these situations. [like if(i < 9) and if( i != 9) ]

What did we change in our pseudocode and algorithm? Why?

We didn’t change anything at the beginning of our pseudocode. We initialized our arrays according to pseudocode.

Here is our changes:

In our pseudocode, we didn’t have control statements that we explained above. So, we add these control statements to our algorithm. [in the method findPlanned() which returns plannedOrderReleases array of the items.]

In our pseudocode, we were finding our plannedOrderReleases and plannedOrderDelivery according to timePhasedNetRequirement, then we changed this part. Now, our algorithm finds plannedOrderRelease and plannnedOrderDelivery according to netRequirements.

When lotSizingRule is 30, 40, 50 or etc except L4L., there are some situations which is need to be controlled. In our pseudocode, we didn’t control all of the possibilities but now we thought every situation and we added some extra control statements.

When lotSizingRule != L4L and netRequirements[i] <=lotSizingRule we arranged our code like this:

plannedRelease[i - leadTime] = Integer.*parseInt*(lotSizingRule);  
plannedDelivery[i] = Integer.*parseInt*(lotSizingRule);  
if ((i + 1) < 10) {  
 amountOnHand[i + 1] = plannedRelease[i - leadTime] - netRequirements[i];  
}

# In our pseuodocode we wrote this statement: amountOnHand[i+leadTime+1] = plannedOrderRelease[i] - timePhased[i]

But we changed this to amountOnHand[i + 1] because we replaced timePhased with netRequirements so the indexes are also changed.

When lotSizingRule != L4L and netRequirements[i] > lotSizingRule, there are two possible cases:

1. When the remainder of division is 0, we added this statement to control it.
2. int y = (netRequirements[i] / Integer.*parseInt*(lotSizingRule));  
   plannedRelease[i - leadTime] = (y \* Integer.*parseInt*(lotSizingRule));  
   plannedDelivery[i] = (y \* Integer.*parseInt*(lotSizingRule));

2.)When the remainder is not equal to zero, we rearranged our code like this:

1. int y = (netRequirements[i] / Integer.*parseInt*(lotSizingRule)) + 1;  
   plannedRelease[i - leadTime] = (y \* Integer.*parseInt*(lotSizingRule));  
   plannedDelivery[i] = (y \* Integer.*parseInt*(lotSizingRule));

Finally,

We removed last part of our pseudo code which is

# end-else-if

# if (plannedOrderRelease[i] is not equal to 0) then

# plannedOrderDelivery[i+leadTime] = plannedOrderRelease[i];

# end-if

We removed this statement because we found the delivery at the same code block where we found plannedOrderReleases. We didn’t want to make our code complicated.

Simple Explanation Of Our Code:

We added a switch case to our code. So the user can select whatever item h/she wants. And our program will show all of the sub-components’ tables which contain 10 week-along order amounts.

To print correct tables of sub-components, we provided the relationship between parent-child items by using objects. [the object of main or objects which have sub-components etc.]

Sub-components’ grossRequirements is equal to parent-components’ plannedOrderReleases. We wrote our algorithm according to this relationship.