Cai Zhongjie

Enhancement of Openbravo POS system

Information and Technology

2010

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Degree Programme of Information Technology

ABSTRACT

Author	Cai Zhongjie
Title	Enhancement of Openbravo POS system
Year	2010
Language	English
Pages	33 Pages + 3 Appendices
Name of Supervisor	Johan Dams

This thesis elaborates the process of the enhancement of Openbravo POS system. The aim of the enhancement is to make the system be able to process the weighted product information correctly and automatically. The enhancement is achieved by changing the original code which is written in JAVA language. After the enhancement, the software is able to process the weighted product information as expected.

Keywords Enhancement, Information process, Automation

Table of Content

1. Introduction

Automation has been more and more widely used in human being's life. For example, without any manual service we can take money from ATM machine, buy drinking from vending machine, buy train ticket from ticket sell machine and so on. It brings convenient to human beings and does fewer mistakes than manual service. Furthermore, it also helps to solve the labour force problem for countries with population aging problem.

In this thesis, the functionality and the enhancement of Openbravo POS, software which is going to bring automation to shop selling, will be fully introduced. The main usage of the software is to save the time for customers buying goods and save cost for shops.

1.1 Point of Sale (POS) System

Point of sale (POS) or checkout is the location where a transaction occurs. A "checkout" refers to a POS terminal or more generally to the hardware and software used for checkouts, the equivalent of an electronic cash register. A POS terminal manages the selling process by a salesperson accessible interface. The same system allows the creation and printing of the receipt.

A wide range of POS applications have been developed on platforms such as Windows and Unix. The availability of local processing power, local data storage, networking, and graphical user interface made it possible to develop flexible and highly functional POS systems. Cost of such systems has also declined, as all the components can now be purchased off-the-shelf. The key requirements that must be met by modern POS systems include: high and consistent operating speed, reliability, ease of use, remote supportability, low cost, and rich functionality.

1.2 Introduction of Openbravo

The openbravo POS is a POS system which supports customers doing self service when they are buying goods in the shop. Customers can choose the goods by themselves, scan the goods information to the machine, and pay money by net bank or credit card. Finally a receipt which lists the information of the products (name, quantity, and price) customer bought will be printed to the customer.

The equipments of the system are established by a barcode reader which gets the product information from scanning the product barcode, a touch screen which allows customer doing operation such as paying money and deleting goods not needed, and a printer which prints the receipt.

The main behaviour of the software is to find the information of the product which has the same barcode of the one scanned by the barcode reader from the database. Then show all the information of the product to the customer on the screen and print the information on the receipt. Of course, the shop as the administrator should set the product information to the database through the system in advance.

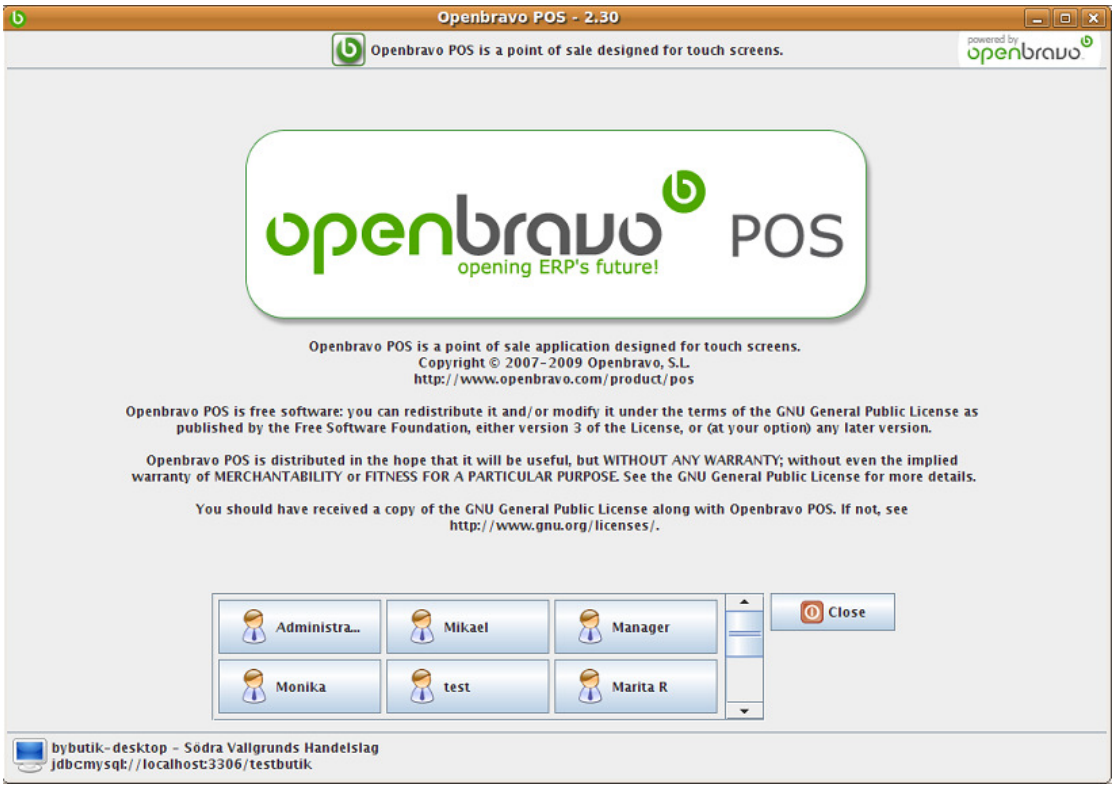

Figure 1 Start Interface

Figure 1 shows the start interface of the software. The shop organizer can login as the administrator with the pass word and the customer can login by clicking their name and inserting the password.

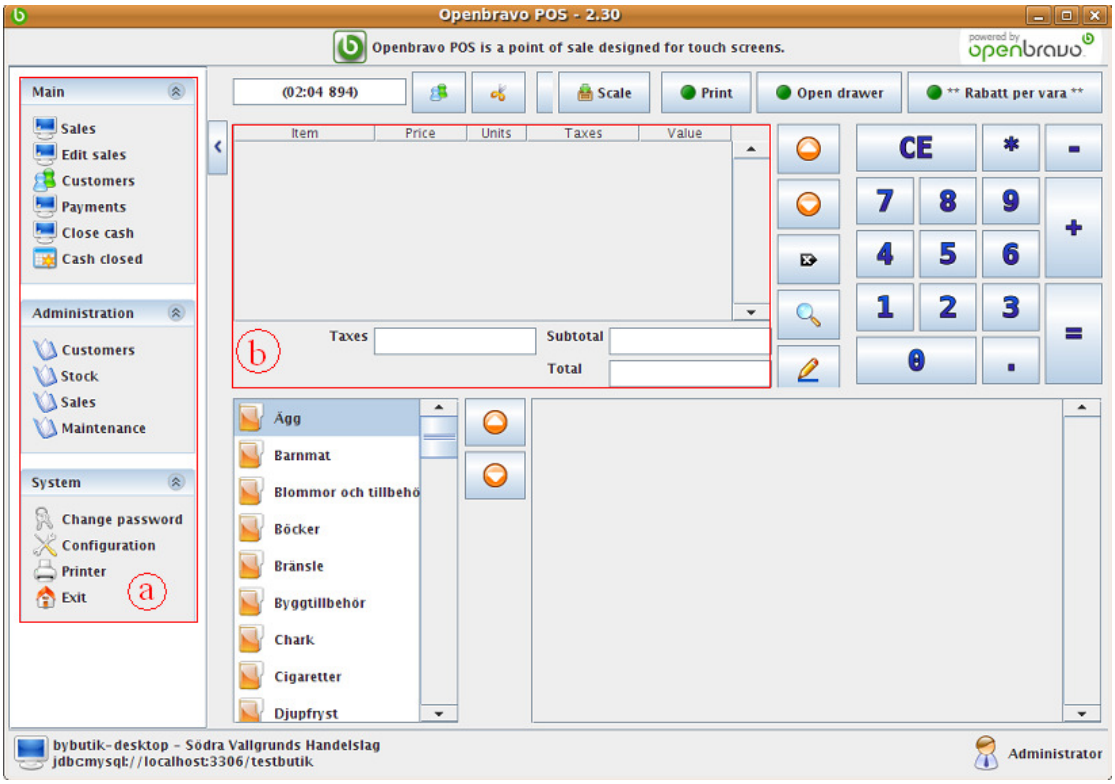

Figure 2 Sales Window

After login, the software shows the window shown as **Figure 2**. It is the user interface of the sales functionality. Customers mainly operate on this window to complete their shopping while the shop mainly user this window to test if their new goods can be recognized by the system.

Block a lists the menu of the software for administrator. The user can change different window for different operation by choosing the options. For customer there only four options which are sales, edit sales, change password and configuration.

Block b is the product information display. The information of the product should be shown on it after the product being scanned. It can only show the information of the products whose barcode can be found in the database.

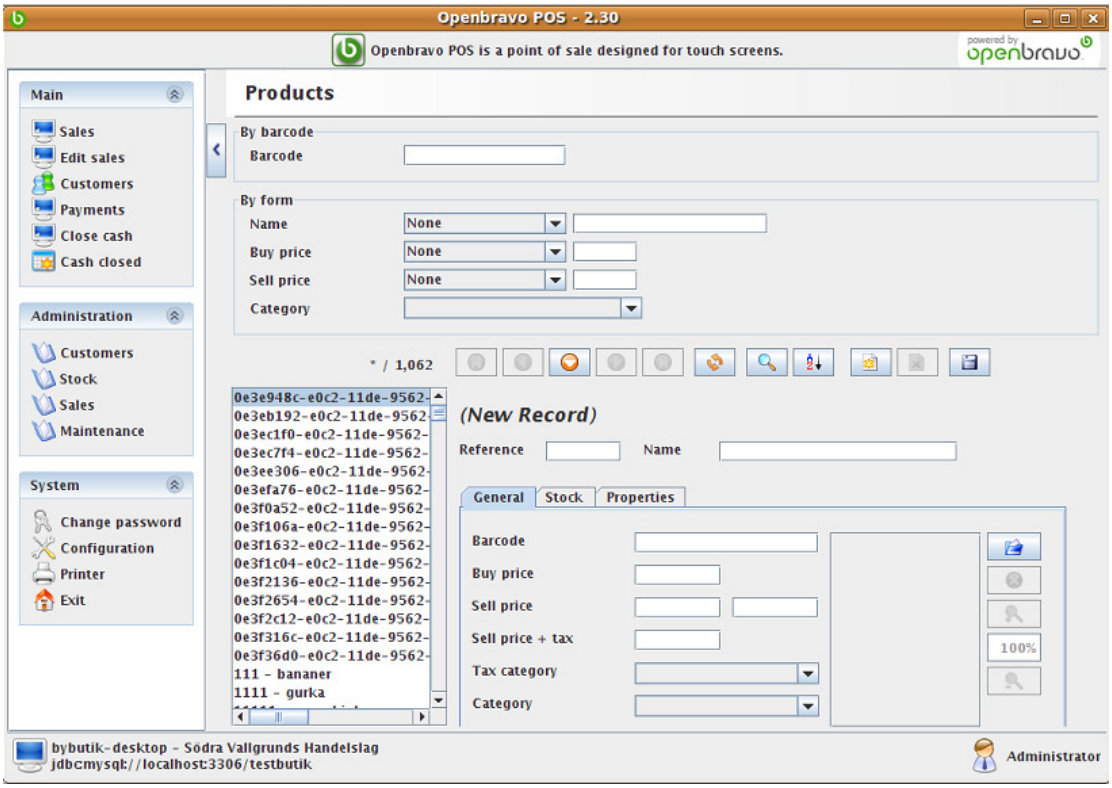

Figure 3 Stock Management Window 1

After entering the stock window from the administrator menu, the administrator can enter the products window by clicking the “Products” button in stock window. This is the main interface for the Stock management. In the window shown as **Figure 3**, the administrator can create a new product or search and edit an existing product. When creating a new product, all the information listed below “New Record” must be given by the administrator.

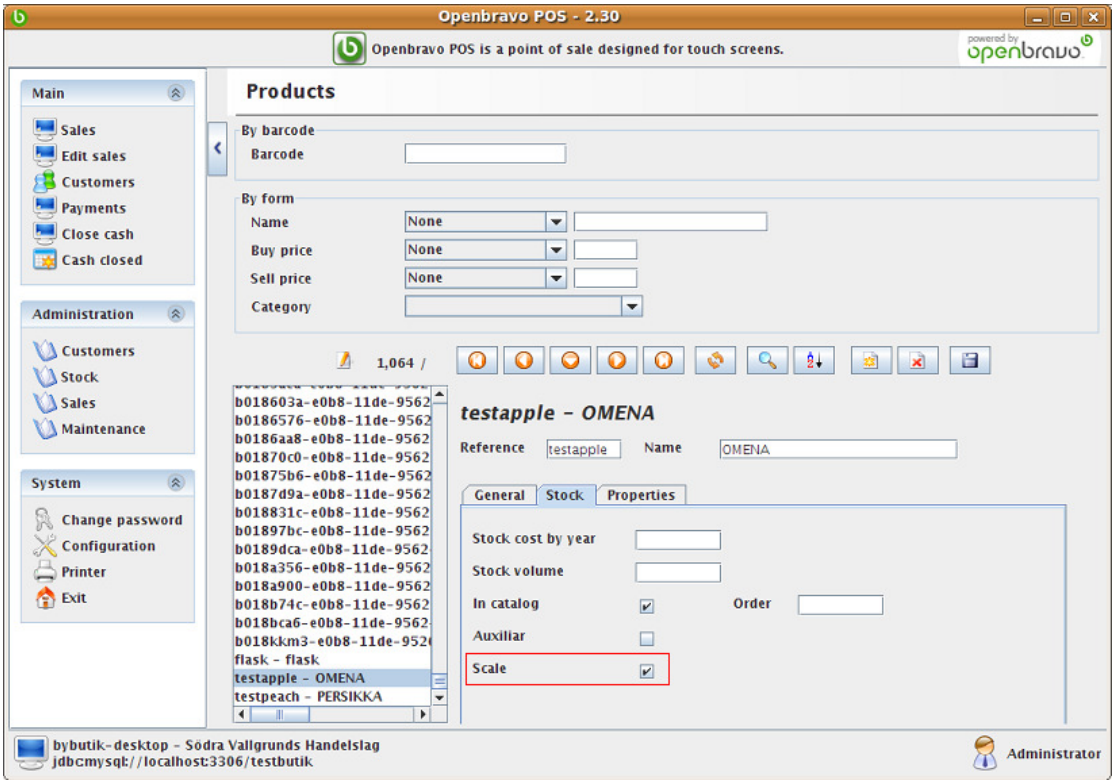

Figure 4 Stock Management Window 2

Figure 4 shows the place to define a product as a weighted product. Under stock label, the shop can tick the scale to make the product as a weighted product when creating a new product. The software will have different behaviour when processing the data of weighted product.

1.3 Introduction of Barcode

A barcode is an optical machine-readable representation of data, which shows certain data on certain products. Originally, barcodes represented data in the widths (lines) and the spaces of parallel lines, and may be referred to as linear or 1D (1 dimensional) barcodes. They also come in patterns of squares, dots, hexagons and other geometric patterns within images termed 2D (2 dimensional) matrix codes. Although 2D systems use symbols other than bars, they are generally referred to as barcodes as well. Barcodes can be read by optical scanners called barcode readers, or scanned from an image by special software.

In point-of-sale management, the use of barcodes can provide very detailed up-to-date information on key aspects of the business, enabling decisions to be made much more quickly and with more confidence. For example:

- Fast-selling items can be identified quickly and automatically reordered to meet consumer demand,
- Slow-selling items can be identified, preventing a build-up of unwanted stock,
- The effects of repositioning a given product within a store can be monitored, allowing fast-moving more profitable items to occupy the best space,
- Historical data can be used to predict seasonal fluctuations very accurately.
- Items may be changed on the shelf to reflect both sale prices and price increases.
- This technology also enables the profiling of individual consumers, typically through a voluntary registration of discount cards. While pitched as a benefit to the consumer, this practice is considered to be potentially dangerous by privacy advocates.

2. Enhancement Requirements Analysis

Normally there are two types of products selling in the shop: packed products and weighted products. For different type of products, the software will have different behaviour of processing the data.

Packed products have the same barcode for the same single product. The shop can set the barcode and the information for them to the database without problem. And the quantity of the product has been taken just depends on how much time the product has been scanned. For example, if a customer bought one box of milk, he needs to scan the milk for one time. If he bought two, then he needs to scan it for tow times. And the system will calculate the price of two box of milk for the customer.

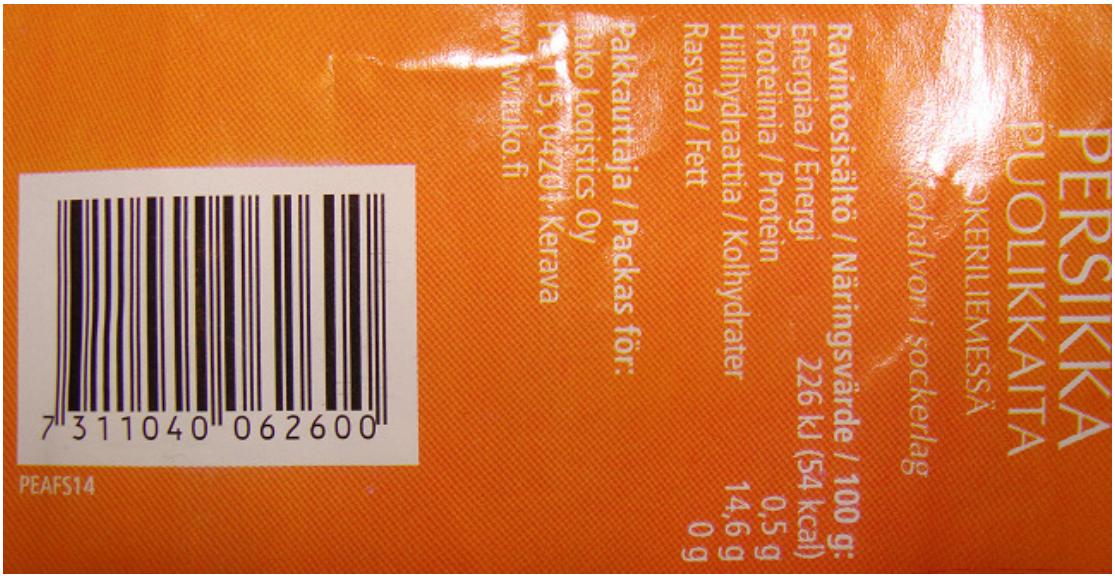

Figure 5 Packed Product Barcode

Figure 5 shows the barcode of a normal packed product, a can of peach. In the market, each can of peach with the same brand is the same weight and has the same

barcode. For this product, the shop can easily set 7311040062600 in the barcode blank when creating the product in database.

The problem appears when setting weighted products. In the shop the quantity of weighted products such as apples and meats can be randomly chosen by the customer. Then the customer takes them to the weighing machine and the machine generates a label with information of price and barcode for the product. The problem is the barcode for the same product changes every time depending on the weight of the product which means it is impossible for the shop to set a fixed barcode for a weighted product to the database. As mentioned before, the system behaviour is based on searching and comparing the barcode. If there is not a fixed barcode for a product, the system will not be able to recognize the product and get the information.

Figure 6 Weighted Product Barcodes

Figure 6 shows two barcodes of the same kind apple. As we can see, the weighing machine generates different barcode because of different weight.

So the first step of the enhancement is to make the shop be able to set a fixed barcode for a weighted product to the system database and the barcode can be a

reference for the system to recognize all the same weighted product no matter how different barcode generated by the weighing machine.

The second step of enhancement is to improve the performance of the software handling the information of products. After the enhancement, the software should be more automation on data processing.

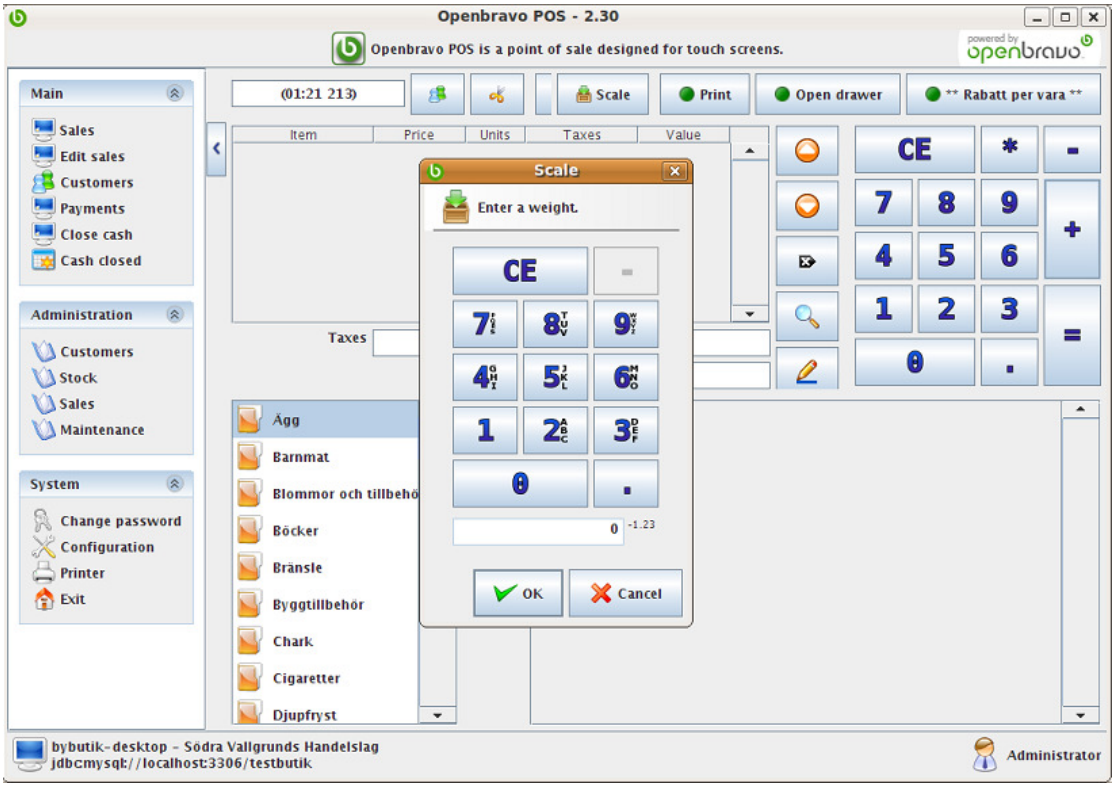

Figure 7 Weight Insert Panel

For the original version of the software, if the product being scanned is the one who has the scale label ticked, there will appears a number insert panel as **Figure 7** Shows. This was designed for customer inserting the weight of the weighted product.

But the shop does not prefer this design. The shop requires the information of weighted products can be shown correctly and automatically in the sales display to the customer. The information should be automatically collected by the software.

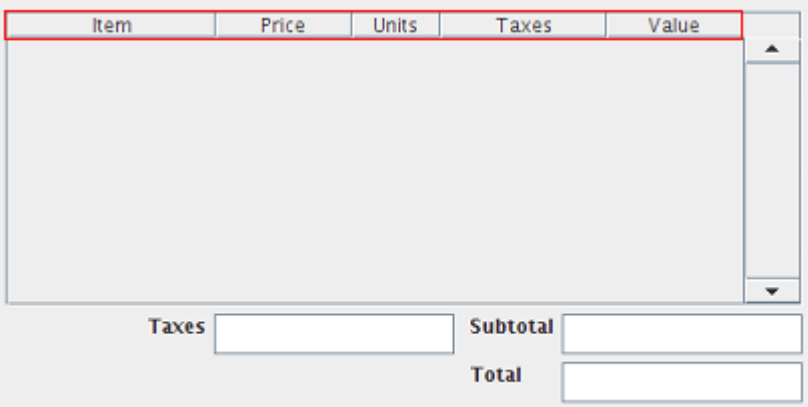

Figure 8 Five Elements of Products' Information

Figure 8 shows the five elements of the products' information. The shop hopes in the sales window the information of the five elements which are item (product name), price (price per kilogram of the weighted product), unit (the weight), taxes (the tax included), and value (the money customer should pay for the product) can be correctly and automatically displayed. Furthermore, these information will be also printed on the receipt.

3. Project Research

3.1 Weighted Product Barcode Research

Since the barcode is the key to solve the problem, it is important to do research of the barcode of the weighted product and find the mathematical regular pattern of the barcode. By further analysis, it can be easily found that the barcode of all the weighted products starts with the same 4-digit number which is '2000'. It is also confirmed by the shop that the weighted product's barcode always starts with '2000'. This characteristic of the barcode is the key for the system to identify if the product is a weighted product or not.

Figure 9 1st-4th Digits of Weighted Product Barcode

Figure 9 shows that the barcodes of apple and banana start with the same number "2000". So, for different weighted products, the first four digits of the barcode are always the same which is "2000".

Continuing with the analysis, it can be found that the 5th-8th digits of the barcode are always the same on the same product. This means the four digits are the code of the product. This characteristic of the barcode can be used to identify different weighted

products for the system and help the shop setting a fixed barcode for a weighted product in the database.

Figure 10 5th-8th Digits of Weighted Product Barcode

Figure 10 shows that the 5th-8th digits of the barcode of apples are the same which is "5157". But the 5th-8th digits of the barcode of banana are different which is "1503".

The rest 5 digits are always changing without mathematical pattern. Actually, the 9th-12th digits represent the final price which corresponding the value of "Value" in the information display. The last digit is a security number which generated by the system randomly.

Figure 11 9th-12th Digits of Weighted Product Barcode

Figure 11 shows no matter what product is, the 9th-12th digits always represent the final price of the product. And the last digit is always changing without any regular pattern.

The conclusion is the barcode should be pre-treatment by the software before comparing to the barcode in the database. The idea is that the shop can set the barcode for a weighted product as '2000XXXX00000' to the database. The 'XXXX' is the 4-digit number that changes according to the product. And on the software side, after getting barcode from the barcode reader, the last 5 digits will be substituted to '00000' by the program at first. Then the barcode will be changed to the format '2000XXXX00000' which is the same format in the database. After that the software will be able to compare and find the same product. On the other hand, the 9th-12th digits are also important for the software calculating the weight of the

product. So it is necessary to store the original barcode in memory before making the substitution.

3.2 Software Research

In order to enhance the system, it is necessary to do research of the old software, get fully understand of the file structure of it and find where the old code is. The enhancement will be implemented based on old code.

Under the directory of the software, we can see there are four folders and ten files (Thumb.db is generated by windows system which is not belong to files of the software.) as **Figure 12** shows. As a JAVA language based program, the most important file is the executable jar file which is the *openbravopos.jar*. It is the main program file.

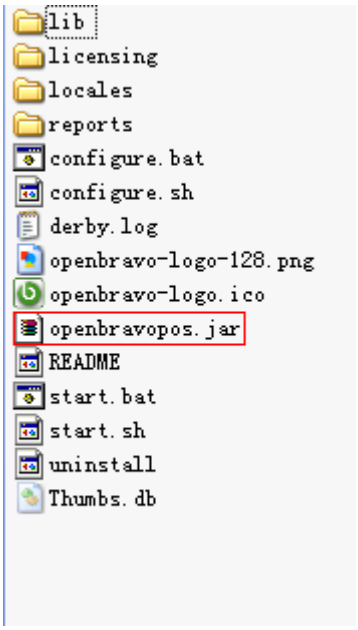

Figure 12 File Structure of Openbravo POS

After opened by unzip software such as WINRAR, it can be found that the *openbravopos.jar* is mainly constituted by many *.class* files. **Figure 13** shows the files in the *openbravopos.jar*.

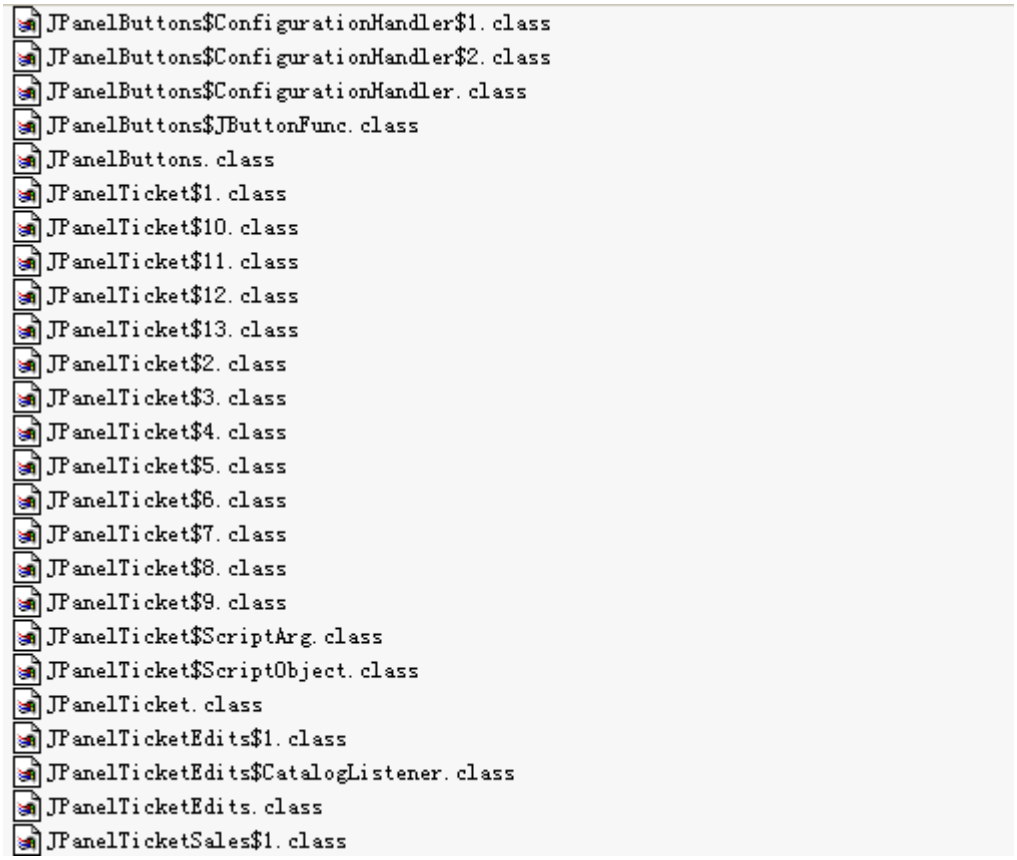

Figure 13 .Class Files in Openbravo.jar

Normally when programming by using JAVA language, a *.java* file is first created by the programmer. After being built, a *.class* file will be generated. The usage of the *.class* file is to translate the content of *.java* file from English to binary number which can be understood by computer.

Now since there are only *.class* files which constitute the software, it is impossible to change the code and enhance the software directly. So the first step of the enhancement is to decompile all the *.class* files which means translate them to *.java* files with normal English codes that human being can understand.

So the conclusion is if we want to develop and enhance the software, we must decompile the software and get the source code at first. And after code implementation, we must create the *.class* file which is used to substitute the old *.class* file in *.jar* file to achieve the enhancement.

4. Development Tools and Environment

- **Decompile Tool**: JD-GUI is a standalone graphical utility that displays Java source codes of *.class* files.

- **Implementation Toll**: Eclipse is the software which is used to build and compiler java source code.

- **Decompressing Tool**: WINRAR is the software for decompressing the *.jar* file and updating the *.class* files inside.

- **Operating System**: Ubuntu Linux operating system and Windows operating system are both used in the process of development. New version of *openbravo.jar* will be made under windows system but it will be tested Linux system, because finally the software will be installed in Linux system and give to the shop.

5. Development Process

Based on the research of the barcode and the software, the process of the enhancement of the system can be made and the steps can be shown as follow:

a) Get the source code by decompiling the .jar file with JD-GUI.
b) Create a new project in Eclipse and add all libraries to the project.
c) Create a *.java* file with the same name of the *.class* file needed to be changed.
d) Copy the source code to *.java* file. Implement the code and build it.
e) Find the new *.class* file. Normally it is under the directory 'workspace/project name/packages/Bin'.
f) Open the *.jar* file with WINRAR. Find and replace the *.class* file which needs to be changed by the new one.
g) Change the *.jar* file of the software by the new one and start the software to test the implementation.

6. Enhancement Design

Before starting with programming code of the enhancement, it is very important to design the enhancement. That means it is necessary to know the behavior of the software and find which part should be enhanced. The best way of design the enhancement is to draw the UML diagrams and see the whole structure and behavior of the system.

6.1 Use Case Diagram

The use case diagram is the diagram which shows the system behavior due to deferent users. **Figure 14** shows the use case diagram of the Openbravo POS system.

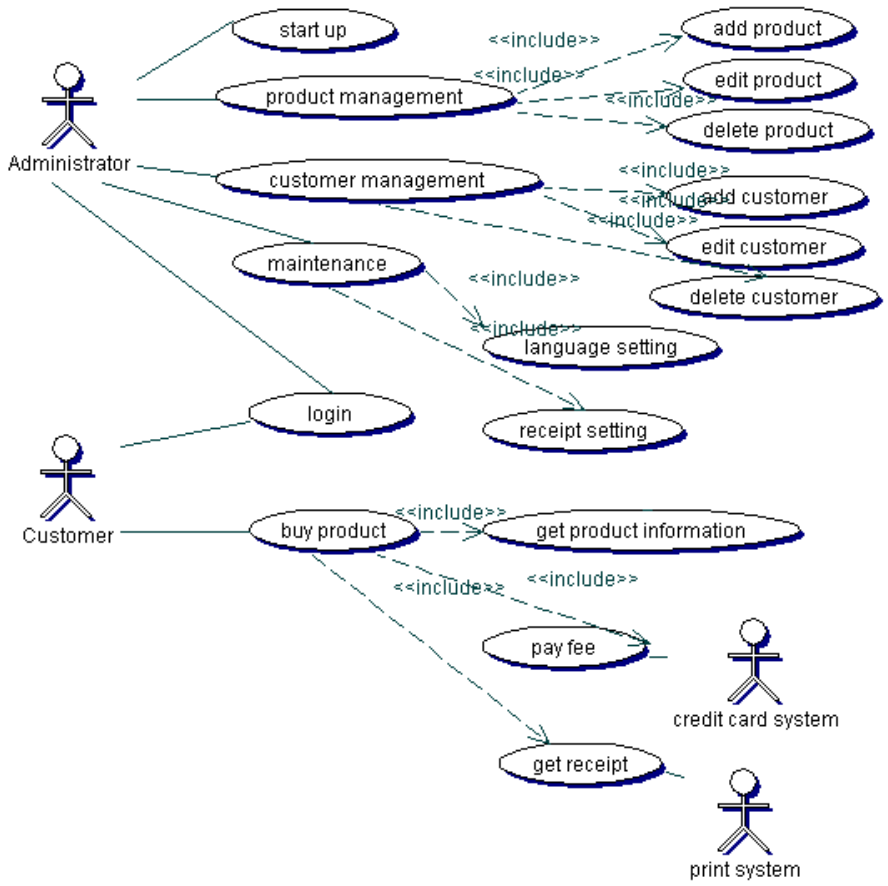

Figure 14 Use Case Diagram

According to the requirements analysis, the problems happens when the administrator add a new weighted product and the customer get information of a weighted product when buy it because the barcode information of weighted product is changeable. By going through the use case diagram, it is quite sure that the enhancement should make the system have normal behavior of adding product and get product information when dealing with a weighted product.

6.2 Sequence Diagram

The sequence diagram is the diagram which shows the sequential logic of the behavior of the system. The diagram describes detail how information and data is processed by the system. According to the requirements analysis, the enhancement will be implemented on behavior of processing the product information. So it is necessary to show the sequence diagram of operation on processing product information for both customer and administrator.

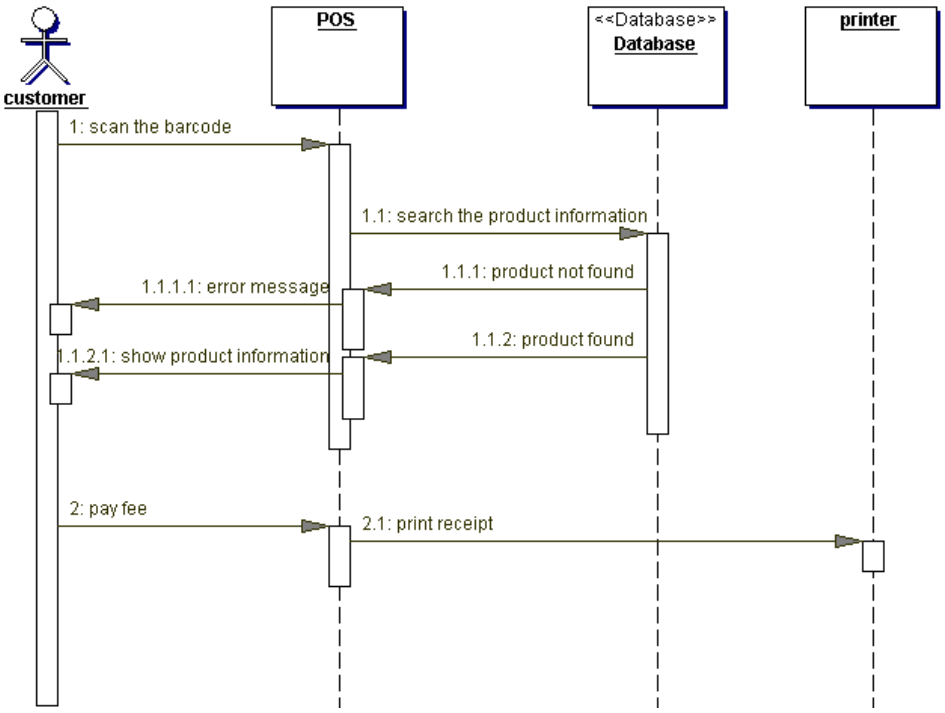

Figure 15 Sequence Diagram of Customer Buying Product

Figure 15 is the sequence diagram of customer when buying products. It can obviously be seen that if a customer wants to buy a product, the system should find the information from the database due to the barcode scanned by the customer. That is the part which the enhancement will be done when a weighted product has been scanned.

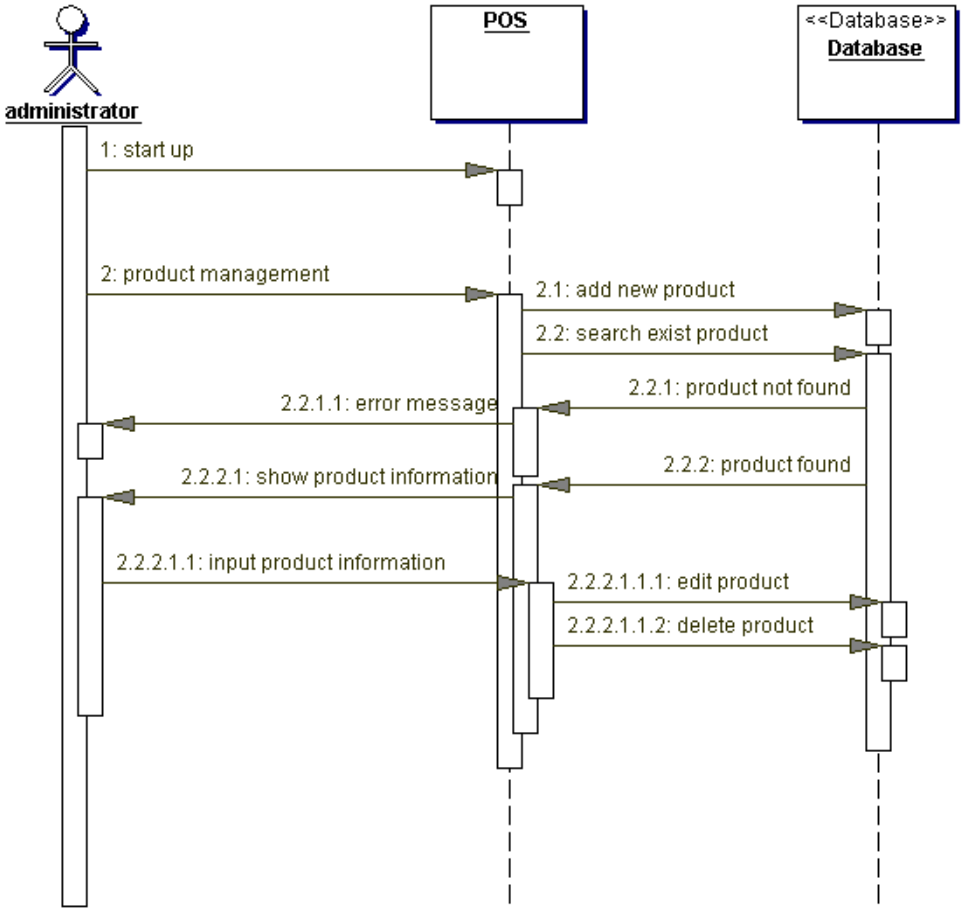

Figure 16 Sequence Diagram of Administrator Managing Product

Figure 16 is the sequence diagram of administrator when managing products. It can obviously be seen that no matter the administrator wants to add or edit a product, the operation is to edit the information in the database. Furthermore the most important part of the information of a product is the barcode which is like the ID of a product.

7. Enhancement Implementation

In this session, the main method in the code which enhancement happened will be shown. New parameters and functions will be created. The implementation will be shown by comparing code before enhanced and after changes have been done.

```
  private void incProductByCode(String paramString)
{
  try
  {
    ProductInfoExt localProductInfoExt = this.dlSales.getProductInfoByCode(paramString);
    if (localProductInfoExt == null)
    {
      Toolkit.getDefaultToolkit().beep();
      new MessageInf(-33554432, AppLocal.getIntString("message.noproduct")).show(this);
      stateToZero();
    }
    else
    {
      incProduct(localProductInfoExt);
    }
  }
  catch (BasicException localBasicException)
  {
    stateToZero();
    new MessageInf(localBasicException).show(this);
  }
}
```

Figure 17 The Original Code of "incProductByCode"

Figure 17 shows the original code of the method "incProductByCode". The main functionality of the method is to get the product information in database by using barcode as reference. It can find and only find packed product which has a fixed barcode.

```
private void incProductByCode(String paramString)
{
  if(paramString.startsWith("2000"))
  {

  try
  {
  String paramStringweight =
  "2000"+paramString.charAt(4)+paramString.charAt(5)+paramString.charAt(6)+paramString.charAt(7)+"00000";
    ProductInfoExt localProductInfoExt = this.dlSales.getProductInfoByCode(paramStringweight);
    if (localProductInfoExt == null)
    {
      Toolkit.getDefaultToolkit().beep();
      new MessageInf(-33554432, AppLocal.getIntString("message.noproduct")).show(this);
      stateToZero();
    }
    else
    {
      incProduct(localProductInfoExt,paramString);
    }
  }
  catch (BasicException localBasicException)
  {
    stateToZero();
    new MessageInf(localBasicException).show(this);
  }
}
else{
      try
      {
        ProductInfoExt localProductInfoExt = this.dlSales.getProductInfoByCode(paramString);
        if (localProductInfoExt == null)
        {
          Toolkit.getDefaultToolkit().beep();
          new MessageInf(-33554432, AppLocal.getIntString("message.noproduct")).show(this);
          stateToZero();
        }
        else
        {
          incProduct(localProductInfoExt,paramString);
        }
      }
      catch (BasicException localBasicException)
      {
        stateToZero();
        new MessageInf(localBasicException).show(this);
      }

}
```

Figure 18 The New Code of "incProductByCode"

Figure 18 shows the new code of the method "incProductByCode" after enhancement. In the method, the barcode of weighted product will be changed to a fixed format "2000XXXX00000" and stored in a new parameter "paramStingweight". After that the original functions will be able be take effect on the barcode of weighted product in fixed format. The first step of the enhancement is implemented.

```
private void incProduct(ProductInfoExt paramProductInfoExt)
{
  if ((paramProductInfoExt.isScale()) && (this.m_App.getDeviceScale().existsScale()))
    try
    {
      Double localDouble = this.m_App.getDeviceScale().readWeight();
      if (localDouble != null)
        incProduct(localDouble.doubleValue(), paramProductInfoExt);
    }
    catch (ScaleException localScaleException)
    {
      Toolkit.getDefaultToolkit().beep();
      new MessageInf(-33554432, AppLocal.getIntString("message.noweight"), localScaleException).show(this);
      stateToZero();
    }
  else
    incProduct(1.0D, paramProductInfoExt);
}
```

Figure 19 The Original Code of "incProduct"

Figure 19 shows the original code of the method "incProduct". This function was designed to get the weight information from the weight inert panel. The weight information is stored in parameter "localDouble".

```
private void incProduct(ProductInfoExt paramProductInfoExt, String paramString)
{
        String PayPrice =
        "0"+paramString.charAt(8)+paramString.charAt(9)+paramString.charAt(10)+paramString.charAt(11);
        Double Price = Double.parseDouble(PayPrice);
        if(paramString.startsWith("20001717")){
            Price = -1 * Price;
        }
        Double unitPrice = paramProductInfoExt.getPriceSell();
        TaxInfo localTaxInfo = this.taxeslogic.getTaxInfo
        (paramProductInfoExt.getTaxCategoryID(), this.m_oTicket.getCustomer());
        Double tax = localTaxInfo.getRate();
        Double unitPriceAfterTax = unitPrice * tax + unitPrice;
        Double unit = Price/unitPriceAfterTax/100;

if ((paramProductInfoExt.isScale()) && (this.m_App.getDeviceScale().existsScale())) {
    Double localDouble = unit;
    if (localDouble != null)
      incProduct(localDouble.doubleValue(), paramProductInfoExt);
} else
  incProduct(1.0D, paramProductInfoExt);

}
```

Figure 20 The New Code of "incProduct"

Figure 20 shows the new code of the method "incProduct" after enhancement. In the method, the "localDouble" gets the weight information no more from the weight insert panel but from the result of calculation. The calculation is based on the information from the barcode and from the database. Finally the weight information will be shown with other information on the sales display panel. The second step of the enhancement is implemented.

8. Testing

In this session, the testing will be done to test if the software has achieved the goal of enhancement. Both packed and weighted products' information will be set into database as the software required. After that the barcode will be scanned to see if the software can process the information correctly and shows it on the screen automatically.

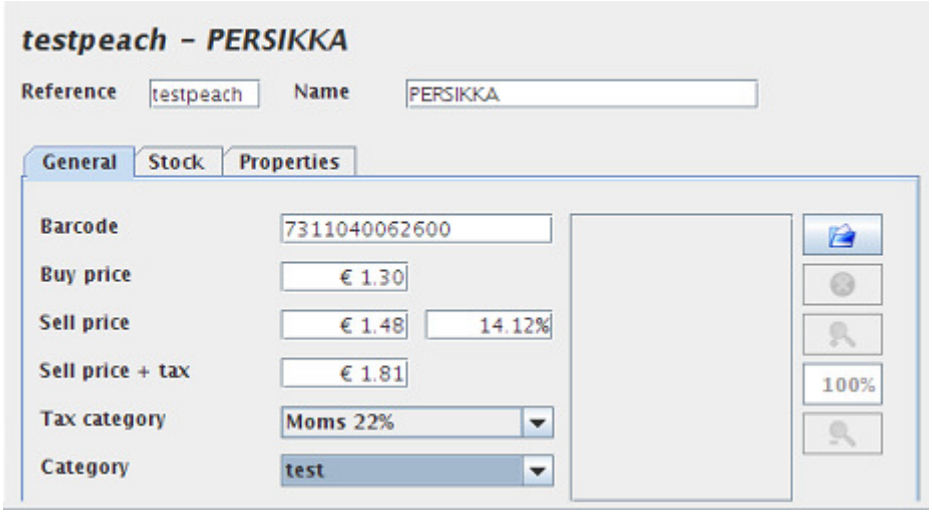

Figure 21 Database Setting of Packed Product

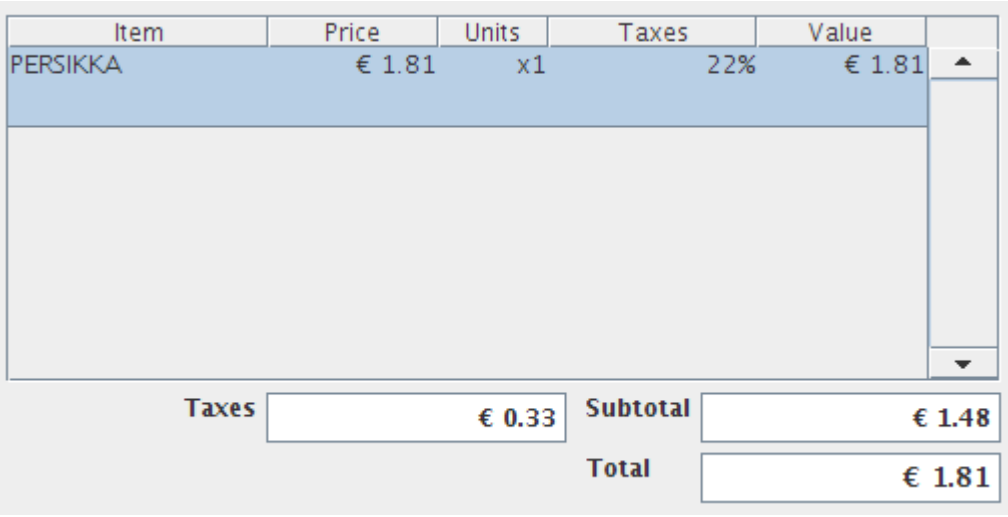

Figure 22 Testing Result of Packed Product

Figure 21 shows the database setting of a packed product and **Figure 22** shows the result after the barcode being scanned. The function works well with the original software. The test is done to make sure that the operation keeps working well after enhancement.

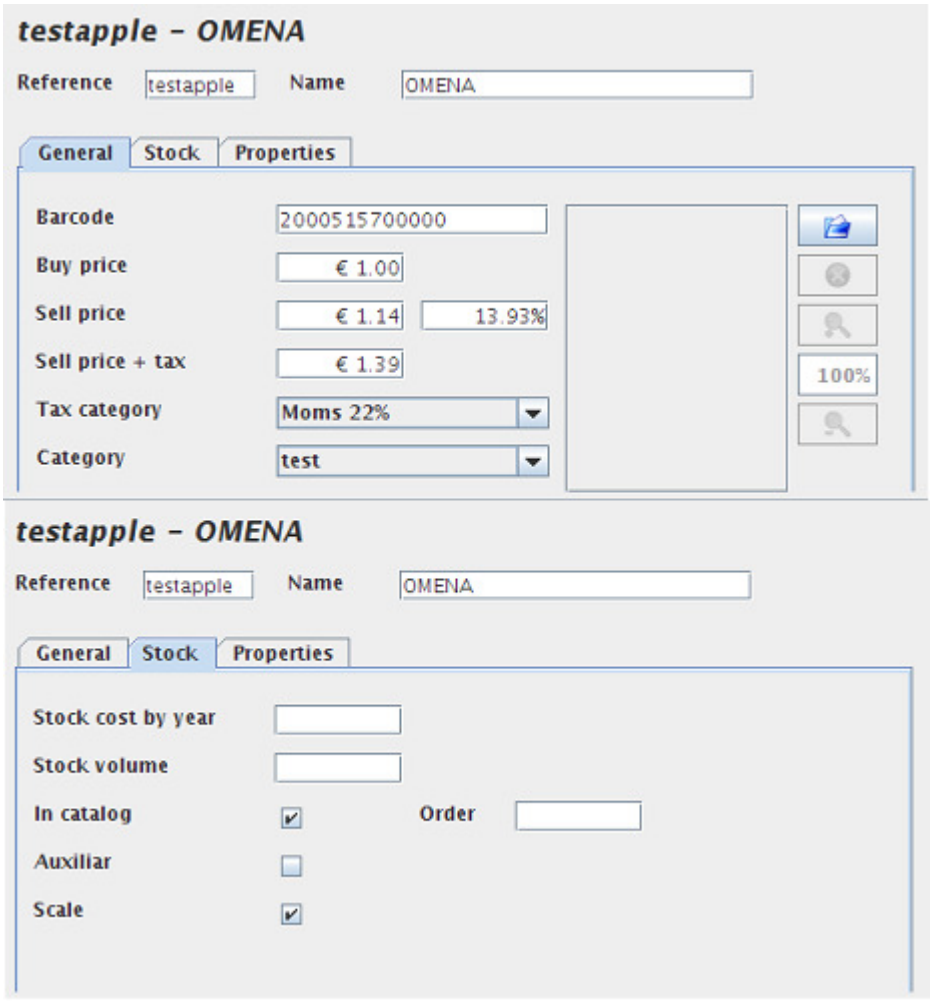

Figure 23 Database Setting of Weighted Product

Figure 23 shows the database setting of a weighted product. Besides the normal information like price and tax, it is important to tick the “Scale” box to set the product as a weighted product in the database.

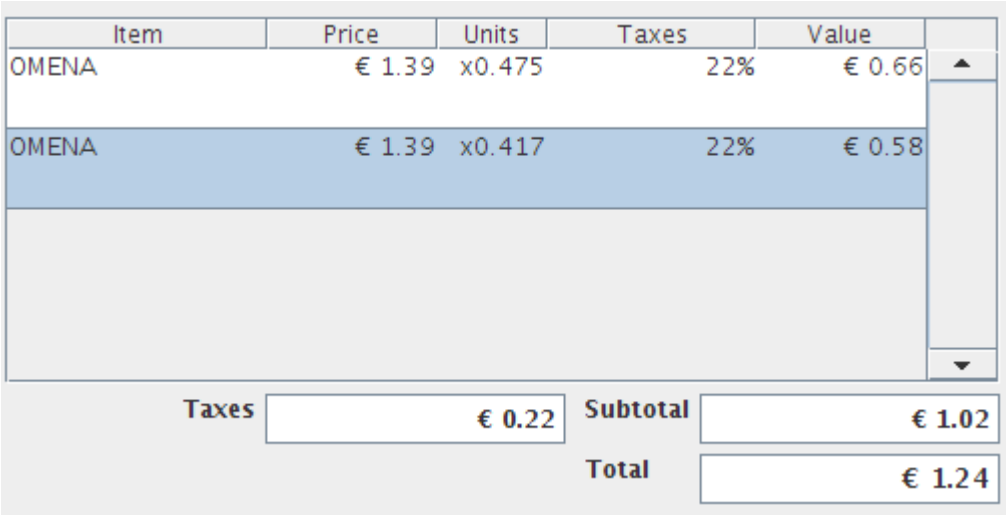

Figure 24 Testing Result of Weighted Product

Figure 24 shows the testing result of weighted product. From the figure it can easily be seen that apples with different weight can be recognized by the system. Furthermore, the information of weight and final price is shown correctly in the display.

The conclusion of the testing is the enhancement has been successfully done. The software now can process the information of both packed and weighted product. The customer only need to scan the barcode and all the operation will be done automatically.

9. References

http://www.openbravo.com/product/pos/

http://en.wikipedia.org/wiki/Barcode

http://en.wikipedia.org/wiki/Point_of_sale

10. Appendices

The appendices is the code of the new version of the software. It can be copied and used for further enhancement.

```
// incProductByCode()


  private void incProductByCode(String paramString)
  {
      if(paramString.startsWith("2000"))
      {


    try
    {
    String paramStringweight =
"2000"+paramString.charAt(4)+paramString.charAt(5)+paramString.charAt
(6)+paramString.charAt(7)+"00000";
      ProductInfoExt localProductInfoExt =
this.dlSales.getProductInfoByCode(paramStringweight);
      if (localProductInfoExt == null)
      {
        Toolkit.getDefaultToolkit().beep();
        new MessageInf(-33554432,
AppLocal.getIntString("message.noproduct")).show(this);
        stateToZero();
      }
      else
      {
        incProduct(localProductInfoExt,paramString);
      }
    }
    catch (BasicException localBasicException)
    {
      stateToZero();
      new MessageInf(localBasicException).show(this);
    }
  }
      else{
           try
```

```
            {
              ProductInfoExt localProductInfoExt =
this.dlSales.getProductInfoByCode(paramString);
              if (localProductInfoExt == null)
              {
                Toolkit.getDefaultToolkit().beep();
                new MessageInf(-33554432,
AppLocal.getIntString("message.noproduct")).show(this);
                stateToZero();
              }
              else
              {
                incProduct(localProductInfoExt,paramString);
              }
            }
            catch (BasicException localBasicException)
            {
              stateToZero();
              new MessageInf(localBasicException).show(this);
            }

      }
  }


//incProduct()


  private void incProduct(ProductInfoExt paramProductInfoExt, String
paramString)
  {

            String PayPrice =
"0"+paramString.charAt(8)+paramString.charAt(9)+paramString.charAt(10
)+paramString.charAt(11);
            Double Price = Double.parseDouble(PayPrice);
            if(paramString.startsWith("20001717")){
                Price = -1 * Price;
            }
            Double unitPrice = paramProductInfoExt.getPriceSell();
            TaxInfo localTaxInfo =
this.taxeslogic.getTaxInfo(paramProductInfoExt.getTaxCategoryID(),
this.m_oTicket.getCustomer());
            Double tax = localTaxInfo.getRate();
```

```
            Double unitPriceAfterTax = unitPrice * tax + unitPrice;
            Double unit = Price/unitPriceAfterTax/100;

   if ((paramProductInfoExt.isScale()) &&
(this.m_App.getDeviceScale().existsScale())) {
       Double localDouble = unit;
       if (localDouble != null)
         incProduct(localDouble.doubleValue(), paramProductInfoExt);
    } else
      incProduct(1.0D, paramProductInfoExt);

  }
```