

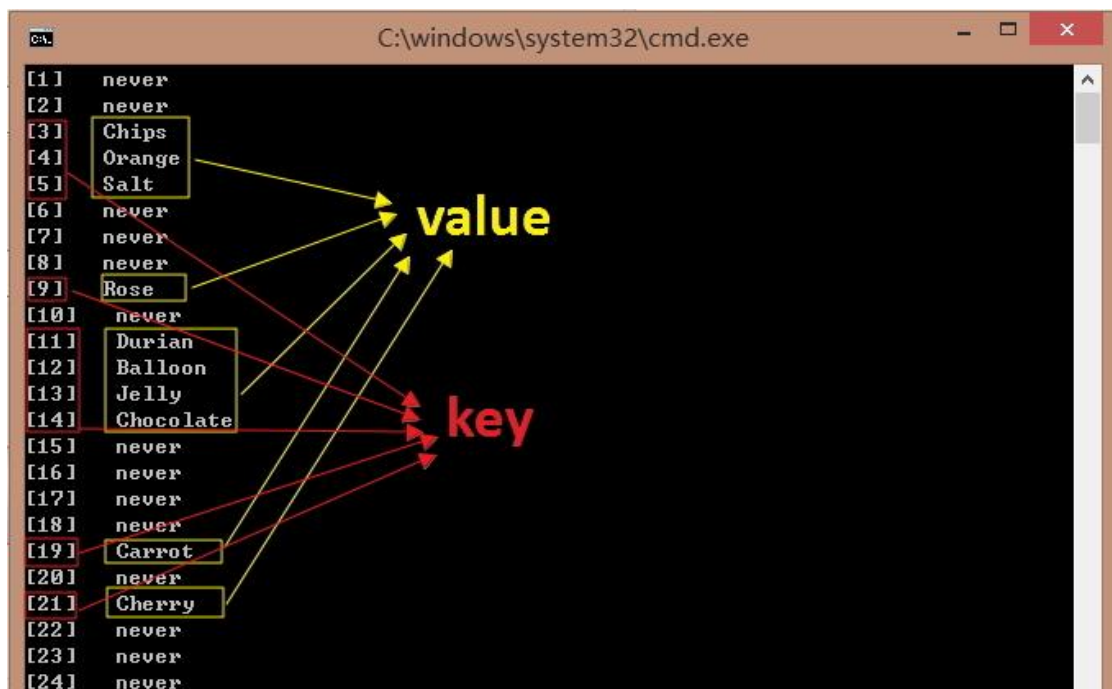
## The following techniques were used to develop the recording system

- Use of a hash table to store information (pages 1-3)
- Use of data files to record information (pages 3-4)
- Use of Java Swing to create Graphical User Interface (pages 5-8)
- Use of Boolean to handle exceptions (pages 9-10)

### Technique 1: Hash table

A hash table is a data structure used to implement an associative array with keys and values. I use this data structure to store all Goods records in Mining's store, because it has a high efficiency in searching,  $O(1)$ . A hash table functions as shown below.

Graph 1



As shown in the above screenshot (Graph 1), in a hash table, every Goods record has its unique value and hence find its corresponding key to store its information. The rest of the keys have a null record indicated "never".

To successfully store a Goods record, the name of the goods needs to be translated into a value and use this value to find its key. It can be achieved by the algorithm below.

Graph 2

```
private int getLoc(String name)
{
    int loc=hashFunction(name);
    if(!table[loc].getName().equals("never")){
        for(int count=0;count<table.length;count++){
            if(table[(loc+probe(count))%COUNT].getName().equals("never")){
                return (loc+probe(count))%COUNT;
            }
        }
    }
    else{
        return loc;
    }
    return -1;
}

private int hashFunction(String name){
    int sum=0;
    for(int count=0;count<name.length();count++){
        char character=name.charAt(count);
        int ascii=(int)character;
        sum=sum+ascii;
    }
    return sum%COUNT;
}

private int probe(int attempt){
    int rtn=(int)(Math.pow((double)attempt,2.0)); //attempt's power of 2. This is the quadratic probing
    return rtn;
}
```

In the algorithm above (Graph 2), method *int getLoc(String name)* is used to translate the parameter *name* into an *int* number value. I define the sum of the ASCII value of each character of *name* to be the value of this Goods record and value % *COUNT* (the length of the hash table which is 100) to be the corresponding key for the value by the method *int hashFunction(String name)*. For example, the value of "Orange" is 79+114+97+110+103+101=604, and its key will be 604%100=4. Therefore the Goods record of "Orange" will be store in the hash table with key of 4 (As shown in Graph 1).

I also use the quadratic probing method if the keys clash with each other. For example, the value of "Salt" is 404 and its key will be 4, crushing with the key of "Orange". At this time, the method *int probe(int attempt)* will return  $1^2=1$  to add up the value to 405 and change the key to 405%100=5. Therefore, the Goods record of "Salt" will be store in the hash table with key of 5 (As shown in Graph 1). If a third Goods has the key of 4, the probing method will return  $2^2=4$  to change the key to 4+4=8. The forth one will have the key  $4+3^2=4+9=13$ , etc.

The advantage of using a hash table should be its high efficiency of searching.

Graph 3

```
private int search(String name) {
    int loc = hashFunction(name);
    int prb;
    for (int count=0; count < table.length; count++) {
        prb = probe(count);
        if (table[(loc+prb) % table.length].getName().equals(name))
            return (loc+prb) % table.length;
        if (table[(loc+prb) % table.length].getName().equals("never"))
            return -1;
    }
    return -1;
}
```

The Big O notation for searching method *int search(String name)* (shown in Graph 3) in a hash table is  $O(1)$ , that is because it can quickly find out the key of a Goods record from its *name* and locate it.

## Technique 2: Data file

I use data file to record down all information about Goods and SalesPerson in Mining's store so that these valuable information will not be volatile or disappear when restarting the recording system. Instead, information can be stored permanently to generate reports.

Graph 4

```
public void writeSalesperson()
{
    try{
        RandomAccessFile f=new RandomAccessFile("Sales Record.dat","rw");
        for(int i=0;i<PERSON;i++){
            String name=record[i].getName();
            if(name.length()>24)
                name=name.substring(0,24);
            f.seek(34*i);
            f.writeUTF(name);
            double sales=record[i].getSales();
            f.seek(34*i+26);
            f.writeDouble(sales);
        }
        f.close();
    }
    catch(IOException e)
    {
        System.out.println(e.getMessage());
    }
}
```

In the screenshot (Graph 4), I use RandomAccessFile to create a data file "Sales Record.dat" that I can read and write on. Each SalesPerson record has

the length of 34 bits, with 26 bits to store its name and 8 bits to store its sales. After creating such file, an icon will appear in the folder:

Graph 5

 MyWindow	2016/12/2 14:32	JAVA
 Ranks	2016/12/25 10:21	JAVA
 Sales Record	2017/1/4 19:24	DAT
 SalesPerson	2016/12/31 22:49	JAVA
 Store	2017/1/4 19:21	JAVA

To read SalesPerson records from the file, I write method *readSalesperson()* (see Graph 6 below):

Graph 6

```
public void readSalesperson() {
    try {
        String name=null;
        double sales=0.0;
        RandomAccessFile f=new RandomAccessFile("Sales Record.dat","r");
        long records=f.length()/34;
        for(int i=0;i<records;i++) {
            f.seek(34*i);
            name=f.readUTF();
            f.seek(34*i+26);
            sales=f.readDouble();
            record[i]=new SalesPerson();
            record[i].set(name,sales);
        }
        f.close();
    } catch(IOException e) {
        System.out.println(e.getMessage());
    }
}
```

At this time, the file will be read only. Find the number of SalesPerson records as I divide the file length by length of a record which is 34. Read out name and sales of each record and store back to the array *record*. In the same way, I use another data file to store the hash table *table*. I use the methods *writeSalesperson()* or *writeTable()* every time *record* or *table* is updated, in order to prevent data loss. When the Recording System is restarted, the array and the hash table will read out the data stored previously from the data files to keep the system running.

### Technique 3: Graphical User Interface (GUI)

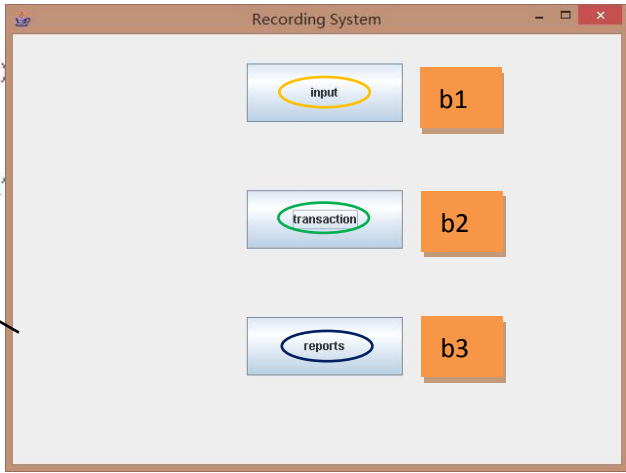
I use Java Swing to develop a Graphical User Interface (GUI) for the Recording System in order to increase its learnability, memorability, and satisfaction for Mining. The increase in usability makes the system more convenient and easier for Mining to use.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class Listener extends JFrame{
    private JButton b1,b2,b3;
    MyWindow f=new MyWindow();

    public Listener(){
        setTitle("Recording System");
        setSize(640,480);
        setLayout(null);

        b1=new JButton("input");
        b2=new JButton("transaction");
        b3=new JButton("reports");
    }
}
```

Imported built-in classes, *Listener* is the first phase of the GUI



I attach *ActionListener* to *JButton* *b1*, *b2*, *b3* (see screenshot below)

```
b1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        f.add1();
        f.setVisible(true);
    }
});

b2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        f.addwhat();
        f.setVisible(false);
    }
});

b3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        f.add3();
        f.setVisible(true);
    }
});
```

Input

Transaction

Report

I will come to class *MyWindow*, the second phase of the GUI, when I click these buttons.

```

MyDoor w=new MyDoor();
public void add1()//two buttons after clicking in
Container c1=getContentPane();
c1.removeAll();
panel.setLayout(null);
panel.add(b4);
panel.add(b5);
b4.setBounds(240,100,160,60);
b5.setBounds(240,250,160,60);
c1.add(panel);
setTitle("input");

public void addwhat(){
w.add2();
w.setVisible(true);
}

public void add3()//two buttons after clicking reports
Container c2=getContentPane();
c2.removeAll();
panel1.setLayout(null);
panel1.add(b6);
panel1.add(b7);
b6.setBounds(240,100,160,60);
b7.setBounds(240,250,160,60);
c2.add(panel1);
setTitle("reports");

```

I also attach *ActionListener* to *JButton* *b4*, *b5*, *b6*, *b7* (see screenshot below)

```

b4.addActionListener(new ActionListener() //Goods
public void actionPerformed(ActionEvent e){
w.add4();
w.setVisible(true);
});

b5.addActionListener(new ActionListener() //Salesperson
public void actionPerformed(ActionEvent e){
w.add5();
w.setVisible(true);
});

b6.addActionListener(new ActionListener() //Goods
public void actionPerformed(ActionEvent e){
w.add6();
w.setVisible(true);
});

b7.addActionListener(new ActionListener() //Salesperson
public void actionPerformed(ActionEvent e){
w.add7();
w.setVisible(true);
});

```



Clicking these buttons, I then come to the final phase of the GUI, *MyDoor*.

```

public void add4() { //4 textfields after clicking goods
    Container c2=getContentPane();
    c2.removeAll();
    panel2.setLayout(null);
    panel2.add(label);
    panel2.add(text);
    panel2.add(label2);
    panel2.add(text2);
    panel2.add(label3);
    panel2.add(text3);
    panel2.add(label4);
    panel2.add(text4);
    panel2.add(save2);
    panel2.add(add1);
    panel2.add(d1);
    label.setBounds(45, 80, 100, 30);
    text.setBounds(145, 80, 150, 30);
    label2.setBounds(345, 80, 100, 30);
    text2.setBounds(445, 80, 150, 30);
    label3.setBounds(45, 180, 100, 30);
    text3.setBounds(145, 180, 150, 30);
    label4.setBounds(345, 180, 100, 30);
    text4.setBounds(445, 180, 150, 30);
    save2.setBounds(440, 300, 160, 60);
    add1.setBounds(40, 300, 160, 60);
    d1.setBounds(240, 300, 160, 60);
    c2.add(panel2);
    setTitle("Goods input");
}

```

```

public void add5() { //2 textfields after clicking salesperson
    Container c3=getContentPane();
    c3.removeAll();
    panel3.setLayout(null);
    panel3.add(label5);
    panel3.add(text5);
    panel3.add(label6);
    panel3.add(text6);
    panel3.add(save3);
    panel3.add(add2);
    panel3.add(d2);
    label5.setBounds(80, 80, 50, 30);
    text5.setBounds(190, 80, 150, 30);
    label6.setBounds(80, 180, 50, 30);
    text6.setBounds(190, 180, 150, 30);
    add2.setBounds(40, 300, 160, 60);
    save3.setBounds(440, 300, 160, 60);
    d2.setBounds(240, 300, 160, 60);
    c3.add(panel3);
    setTitle("Salesperson input");
}

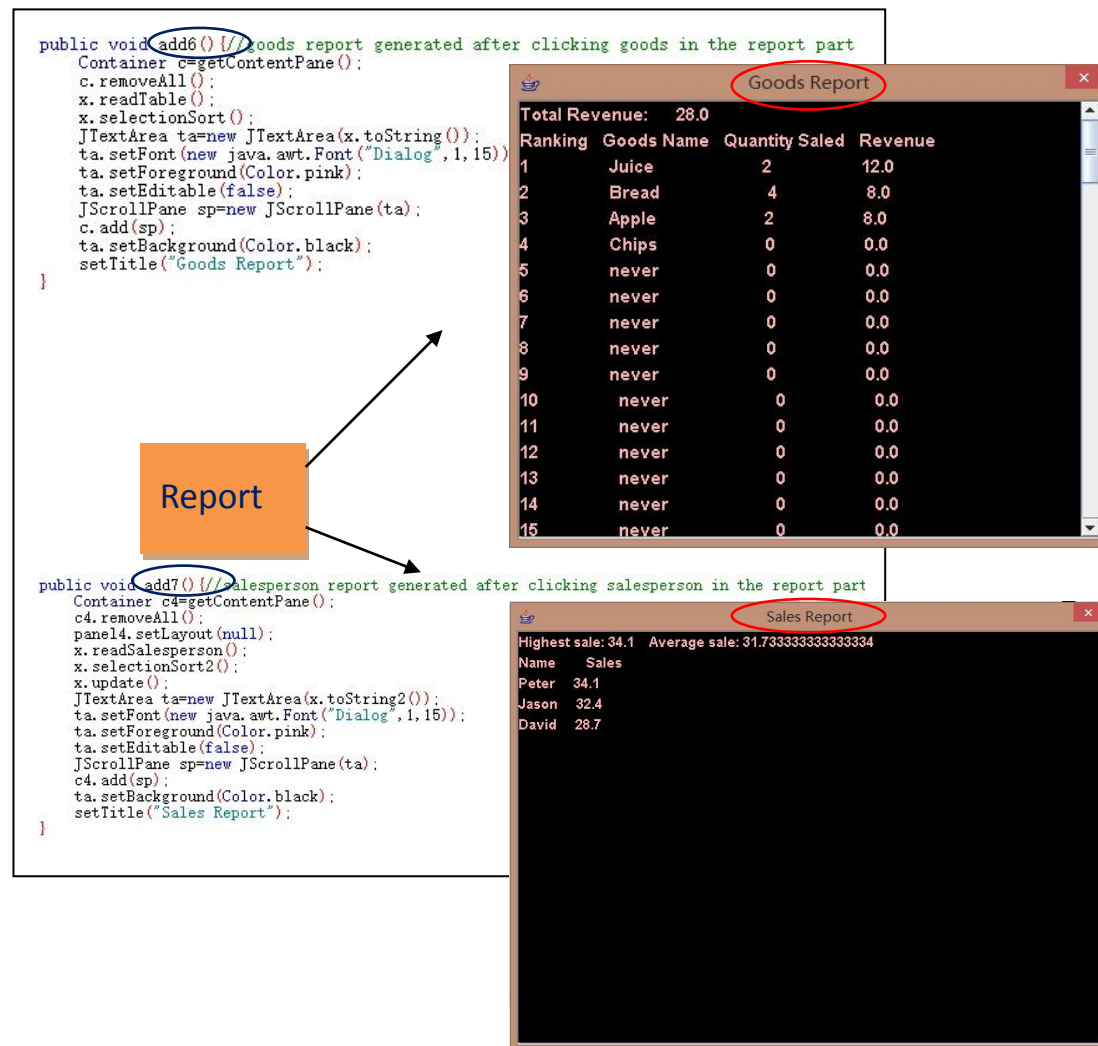
```

```

public void add2() { //3 textfields after clicking transaction
    Container c1=getContentPane();
    c1.removeAll();
    panel1.setLayout(null);
    panel1.add(label7);
    panel1.add(text7);
    panel1.add(label8);
    panel1.add(text8);
    panel1.add(label9);
    panel1.add(text9);
    panel1.add(save1);
    panel1.add(add3);
    label7.setBounds(40, 120, 70, 30);
    text7.setBounds(110, 120, 100, 30);
    label8.setBounds(220, 120, 70, 30);
    text8.setBounds(290, 120, 100, 30);
    label9.setBounds(400, 120, 80, 30);
    text9.setBounds(480, 120, 100, 30);
    add3.setBounds(140, 300, 160, 60);
    save1.setBounds(340, 300, 160, 60);
    c1.add(panel1);
    setTitle("Transaction");
}

```

The final part of the recording system is the Report section. I choose cannot-be-edited JTextArea (see below) to be the platform for the reports of Goods and SalesPerson.

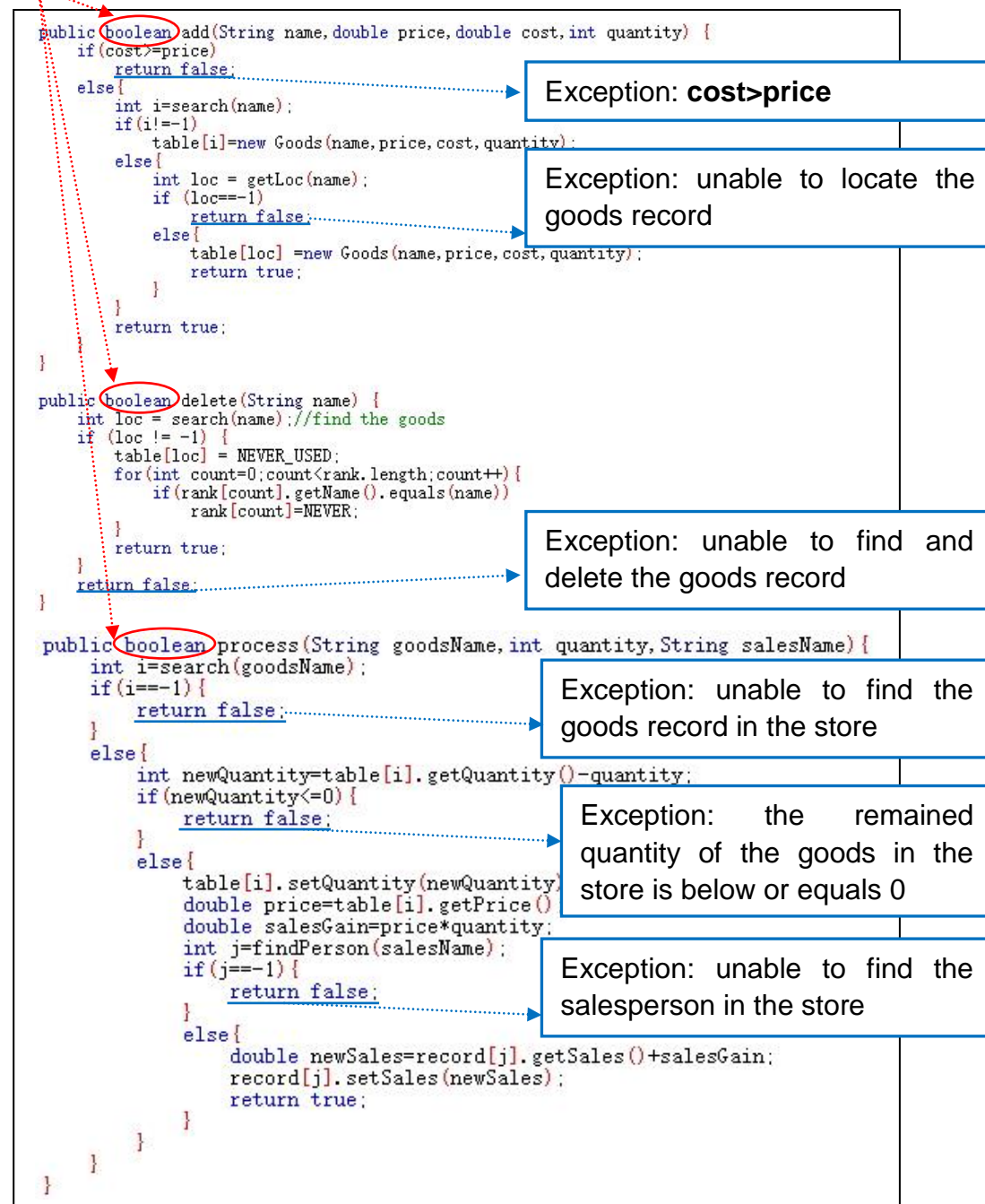




## Technique 4: Boolean

In this recording system, I design Boolean to handle exceptions that may occur during process. For example, if the cost of a product is higher than its price, this is an abnormal situation and needs exception handling to prevent system breakdown. Thus exception handling increases the system's fault tolerance and usability, and thus guarantees Mining to run the Recording System smoothly.

First, set methods with return type of boolean instead of void methods



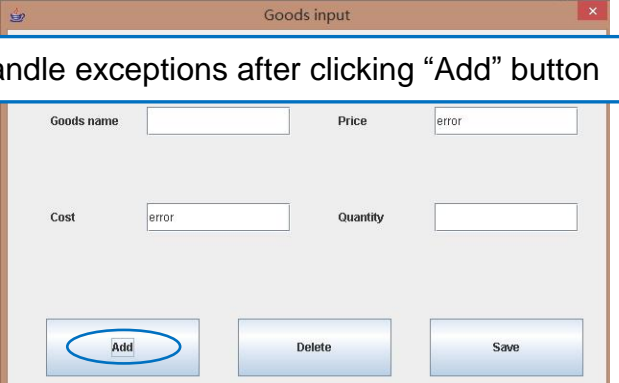
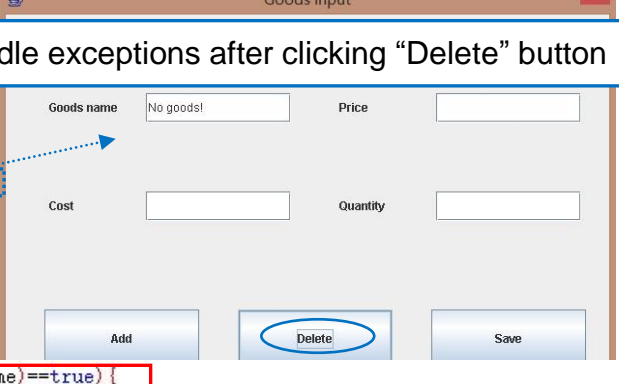
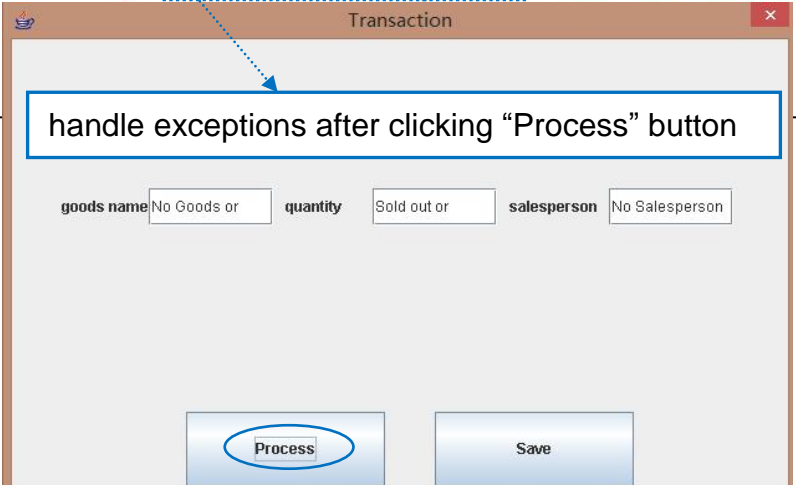
Then use the **if** loop to take the value of Boolean to decide the actions afterward

```
add1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String name=text.getText();
        double price=Double.parseDouble(text2.getText());
        double cost=Double.parseDouble(text3.getText());
        int quantity=Integer.parseInt(text4.getText());
        if(x.add(name,price,cost,quantity)==true) //successfully :
            text.setText("");
            text2.setText("");
            text3.setText("");
            text4.setText("");
        else{
            text.setText("");
            text2.setText("error");
            text3.setText("error");
            text4.setText("");
        }
    }
});

save2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        x.writeTable();
    }
});

d1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String name=text.getText();
        if(x.delete(name)==true) //successfully :
            text.setText("");
        else
            text.setText("No goods");
    }
});

add3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String name=text7.getText();
        int quantity=Integer.parseInt(text8.getText());
        String sName=text9.getText();
        if(x.process(name,quantity,sName)==true) {
            text7.setText("");
            text8.setText("");
            text9.setText("");
        }
        else{
            text7.setText("No Goods or"); //cannot find this Goods in the hash table
            text8.setText("Sold out or"); //Goods' quantity equals or below 0
            text9.setText("No Salesperson"); // cannot find this salesperson in the record
        }
    }
});
```

(Word count: 937 words)