



# WAREHOUSE APPLICATION

*OPTIMISING THE ITEM ENTRY AND  
WITHDRAWAL PROCESSES*



## IB COMPUTER SCIENCE DOSSIER HIGHER LEVEL

**STUDENT** : ALVARO MORALES  
**CANDIDATE N°** : D-00633-032  
**TEACHER** : Mr. Jason Silby  
**SESSION** : November 2010  
**DATE** : August 2010

Alvaro Morales  
Candidate Number: D-000633-032

---

## **ACKNOWLEDGEMENTS**

---

To Mr. Silby, my Computer Science teacher, for his constant support and advice throughout this long project.

And to my user, Sr. Santiago Lock at Cementos Lima, special thanks for his attention and guidance.

---

## TABLE OF CONTENTS

---

<b>ANALYSIS .....</b>	<b>9</b>
<b>A1 – ANALYSING THE PROBLEM .....</b>	<b>10</b>
<i>The Warehouse.....</i>	10
<i>The Existing Process.....</i>	10
<i>The Current System.....</i>	15
<i>Problems and Shortcomings .....</i>	19
<i>Layout of Warehouse 1.....</i>	21
<b>A2 – CRITERIA FOR SUCCESS.....</b>	<b>25</b>
<b>A3 – PROTOTYPE SOLUTION.....</b>	<b>28</b>
<i>Initial Design .....</i>	29
<i>Login Screen .....</i>	31
<i>Main Screen .....</i>	31
<i>Navigation .....</i>	32
<i>Dialogs .....</i>	35
<i>Account Settings .....</i>	36
<i>Manage Users .....</i>	36
<i>Create a New Item .....</i>	37
<i>Item Enquiry.....</i>	39
<i>Item Description .....</i>	40
<i>Multiple Item Search .....</i>	44
<i>Item Entry.....</i>	46
<i>Item Exit Process .....</i>	47
<b>DETAILED DESIGN.....</b>	<b>50</b>
<b>B1 – DATA STRUCTURES.....</b>	<b>51</b>
<i>Users .....</i>	51
<i>Items.....</i>	52
<i>Transactions.....</i>	59
<i>Files.....</i>	62
<b>B2 - ALGORITHMS.....</b>	<b>66</b>

<i>Algorithm list</i> .....	66
<i>User management</i> .....	68
<i>Item Handling</i> .....	72
<i>Operations of the Binary Tree of Item Indexes</i> .....	86
<i>Transactions Handling</i> .....	94
<i>Operations of the Linked List of TransactionRecords</i> .....	96
<b>B3 – MODULAR DESIGN</b> .....	<b>100</b>
<b>THE PROGRAM</b> .....	<b>104</b>
<b>C1 – PROGRAM LISTING</b> .....	<b>105</b>
<i>Mastery Achieved</i> .....	105
<b>C2 –HANDLING ERRORS</b> .....	<b>417</b>
<i>AccountSettings</i> .....	419
<i>CreateNewItem</i> .....	420
<i>EditUser</i> .....	425
<i>Index</i> .....	427
<i>IndexBTree</i> .....	428
<i>Item</i> .....	431
<i>ItemDescription</i> .....	432
<i>ItemEntry</i> .....	435
<i>ItemExit</i> .....	436
<i>Items</i> .....	438
<i>ItemSearch</i> .....	439
<i>Location</i> .....	440
<i>LoginScreen</i> .....	441
<i>MainScreen</i> .....	443
<i>ManageUsers</i> .....	444
<i>MultipleItemSearch</i> .....	446
<i>MultipleSearchResultsTableModel</i> .....	448
<i>NewUser</i> .....	449
<i>SearchResultsTableModel</i> .....	451
<i>TransactionList</i> .....	452

<i>TransactionProcessingTableModel</i> .....	453
<i>TransactionRecord</i> .....	453
<i>Transactions</i> .....	454
<i>TransactionsTableModel</i> .....	455
<b>C3 – SUCCESS OF THE PROGRAM .....</b>	<b>457</b>
<b>DOCUMENTATION .....</b>	<b>459</b>
<b>D1 – TEST OUTPUT.....</b>	<b>460</b>
<i>Login</i> .....	460
<i>Loading Appropriate User permissions</i> .....	465
<i>Loading The Account Settings Screen</i> .....	469
<i>Change User's Password</i> .....	470
<i>Loading the Manage Users screen</i> .....	472
<i>Selecting a User</i> .....	473
<i>Loading the New User Screen</i> .....	474
<i>Creating A New User</i> .....	475
<i>Delete A User</i> .....	480
<i>Loading the Edit User Screen</i> .....	483
<i>Editing A User</i> .....	485
<i>Loading the Create a New Item Screen</i> .....	490
<i>Creating a New Item</i> .....	492
<i>Loading the Item Search Screen</i> .....	512
<i>Searching for an Item by Code</i> .....	514
<i>Searching for an Item by Partial Name</i> .....	520
<i>Searching for an Item by Exact name</i> .....	525
<i>Searching for an Item By Group</i> .....	530
<i>Searching for an Item by Location</i> .....	533
<i>Loading the Item Description Screen</i> .....	536
<i>Editing Item Information</i> .....	537
<i>Viewing Item Image</i> .....	552
<i>Changing an Item's Image</i> .....	554
<i>Viewing Item Transactions</i> .....	558

<i>Deleting a Transaction .....</i>	563
<i>Deleting an Item .....</i>	571
<i>Creating a New Item in a file with Spaces.....</i>	577
<i>Loading the Multiple Item Search Screen .....</i>	580
<i>Searching for One or More items by Code.....</i>	583
<i>Searching for One or More Items by Partial Name.....</i>	590
<i>Searching For One or More Items By Exact Name .....</i>	595
<i>Searching For One or More Items by Group.....</i>	600
<i>Searching For One or More items by Location.....</i>	603
<i>Adding One or More Items to the List of Items to Process .....</i>	606
<i>Removing One or More Items from the List of Items to Process .....</i>	612
<i>Processing an Item Entry Transaction.....</i>	615
<i>Processing an Item Exit Transaction .....</i>	626
<i>Getting the Optimal Pickup Route .....</i>	639
<b>D2 – EVALUATING SOLUTIONS .....</b>	<b>643</b>

---

## INTRODUCTION

---

Cementos Lima S.A. is Peru's largest and most important cement company. The company was founded in 1967 after the merging of several other factories, and has steadily grown to be the corporation that it is today.

The identified end-user for my Computer Science Internal Assessment is Mr. Santiago Lock, the Head of the Systems Department at Cementos Lima. He has been working at the company for over 20 years, and has supervised and pioneered the shift from paper-based tasks to computer-based operations at the corporation.

In addition, valuable consulting help was given to me by Mr. Jaime Salazar, the Head of Warehouse Operation and Maintenance and his staff.



## Section A

---

# Analysis

---

## A1 – Analysing the Problem

---

### THE WAREHOUSE

Cementos Lima has 4 main warehouses that store things ranging from 10-tonne machines to 50gr bolts and nails. These facilities are huge. The company is audited once a year following legal requirements, and needs to have up-to-date figures of what they store for the numbers to add up correctly. By January 2010, the total value of the stock was S/. 45,444,483, roughly over USD\$16 million.

The warehouse is essential for production at the company. If certain products are not available at any given time, such as heat-resistant bricks to pat the inside of the main oven, production could halt altogether. Roughly calculated, Cementos Lima S.A. sells about \$1 million US a day, so even if the ovens would stop working for an hour, losses would be substantial.

This all translates to the need of a fast, efficient item rotation scheme, where items are requested and delivered in the least amount of time possible, and a proper and reliable inventory of stock is kept updated at all times.

### THE EXISTING PROCESS

Cementos Lima developed a suite of applications under the name of "CeliSistemas" back in the mid-1990s. Programs are specialized and address all of the company's needs, including accounting, purchases, sales, etc. The bundle includes a Warehouse system that keeps track of entries and exits of products in the warehouse.

In an interview with Mr. Lock, dated February 26<sup>th</sup> 2010, we discussed the current processes and tasks related to the warehouse. The process goes as follows:

#### **The Request**

A company worker requires an item. The first step is to submit a request, called the "*Solicitud de Adquisición de Materiales*" or Purchase of Materials Request (Fig. 1.1).

This request is submitted electronically via the warehouse system, containing information such as Product name, quantity, family of product. The system logs the name of the user requesting the product, and asks for the company division that the products will be for (e.g. Systems).

### Fig. 1.1 – Purchase of Materials Request

Includes general information such as solicitor, department where materials will be used, type of item. Includes product-specific information like item name, description, quantity, price. Physical copy signed by relevant authorities; electronic copy shows who it has been signed by.

 **Cementos Lima S.A.**

Pag. 1 de 1

**SOLICITUD DE ADQUISICION NRO. 20100166**

Tipo de Pedido :	LOCAL		Fecha :	03/02/2010																																																																																																																																					
Solicitante :	0091	LOCK S.	Prioridad :	Normal																																																																																																																																					
Area origen :	1220	SISTEMAS	Solicitud de Materiales Criticos : <input type="radio"/> Si <input checked="" type="radio"/> No																																																																																																																																						
Area Destino :	1220	SISTEMAS	Situacion : Material ingresado al Kardex (Atendido)																																																																																																																																						
Centro Costo :			Para ser usado en : Forma Inmediata																																																																																																																																						
ADI :	8125101	Hardware																																																																																																																																							
Maquina :																																																																																																																																									
Referencias :																																																																																																																																									
DESTINO: ITEM 1.- GG. ITEMS 2 Y 3.-SIS.																																																																																																																																									
OBS.- SE ADJUNTA COTIZACION DE OMNIA SOLUTION Y DE EMPORIUM IBEROAMERICANA DE SISTEMAS.																																																																																																																																									
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Nro</th> <th>Material</th> <th>Actuales Cantidad</th> <th>Valores</th> <th>Ultimo Pedido Número</th> <th>Fecha</th> <th>Cantidad</th> <th>Consumo. Ult 12. Meses</th> <th>Solicitado</th> <th>Und.</th> <th>V.Unit.Ref US\$</th> <th>Valor Ref US\$</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>ELTN. 224599</td> <td>0.00</td> <td>0.00</td> <td>20091393, 2</td> <td>27/08/2009</td> <td>1.06</td> <td>2.00 U</td> <td>1.00 U</td> <td>\$</td> <td>58.00</td> <td>58.00</td> </tr> <tr> <td colspan="12">DVD RW EXT SAMSUNG SLIM NEGRO 22X NRO PARTE.-SE-S084C/RSBN</td> </tr> <tr> <td colspan="12">Obs.: GG Dest.: ADI: 8125101 Máquina: -</td> </tr> <tr> <td>2</td> <td>ELTN. 234101</td> <td>0.00</td> <td>0.00</td> <td>20090782, 7</td> <td>07/05/2009</td> <td>25.06</td> <td>55.00 U</td> <td>25.00 U</td> <td>\$</td> <td>62.61</td> <td>1,565.25</td> </tr> <tr> <td colspan="12">CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A</td> </tr> <tr> <td colspan="12">Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -</td> </tr> <tr> <td>3</td> <td>ELTN. 223498</td> <td>0.00</td> <td>0.00</td> <td>20091521, 8</td> <td>24/09/2009</td> <td>6.06</td> <td>12.00 U</td> <td>5.00 U</td> <td>\$</td> <td>13.00</td> <td>65.00</td> </tr> <tr> <td colspan="12">MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL</td> </tr> <tr> <td colspan="12">Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -</td> </tr> <tr> <td colspan="6">Esta Solicitud carece de valor sin la autorización indicada líneas adelante</td> <td colspan="6">Valor Referencial US \$ 1,688.25</td> </tr> </tbody> </table>						Nro	Material	Actuales Cantidad	Valores	Ultimo Pedido Número	Fecha	Cantidad	Consumo. Ult 12. Meses	Solicitado	Und.	V.Unit.Ref US\$	Valor Ref US\$	1	ELTN. 224599	0.00	0.00	20091393, 2	27/08/2009	1.06	2.00 U	1.00 U	\$	58.00	58.00	DVD RW EXT SAMSUNG SLIM NEGRO 22X NRO PARTE.-SE-S084C/RSBN												Obs.: GG Dest.: ADI: 8125101 Máquina: -												2	ELTN. 234101	0.00	0.00	20090782, 7	07/05/2009	25.06	55.00 U	25.00 U	\$	62.61	1,565.25	CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A												Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -												3	ELTN. 223498	0.00	0.00	20091521, 8	24/09/2009	6.06	12.00 U	5.00 U	\$	13.00	65.00	MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL												Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -												Esta Solicitud carece de valor sin la autorización indicada líneas adelante						Valor Referencial US \$ 1,688.25					
Nro	Material	Actuales Cantidad	Valores	Ultimo Pedido Número	Fecha	Cantidad	Consumo. Ult 12. Meses	Solicitado	Und.	V.Unit.Ref US\$	Valor Ref US\$																																																																																																																														
1	ELTN. 224599	0.00	0.00	20091393, 2	27/08/2009	1.06	2.00 U	1.00 U	\$	58.00	58.00																																																																																																																														
DVD RW EXT SAMSUNG SLIM NEGRO 22X NRO PARTE.-SE-S084C/RSBN																																																																																																																																									
Obs.: GG Dest.: ADI: 8125101 Máquina: -																																																																																																																																									
2	ELTN. 234101	0.00	0.00	20090782, 7	07/05/2009	25.06	55.00 U	25.00 U	\$	62.61	1,565.25																																																																																																																														
CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A																																																																																																																																									
Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -																																																																																																																																									
3	ELTN. 223498	0.00	0.00	20091521, 8	24/09/2009	6.06	12.00 U	5.00 U	\$	13.00	65.00																																																																																																																														
MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL																																																																																																																																									
Obs.: SIS□□CC 91504 OF.11021 Dest.: ADI: Máquina: -																																																																																																																																									
Esta Solicitud carece de valor sin la autorización indicada líneas adelante						Valor Referencial US \$ 1,688.25																																																																																																																																			

1/6 ROJAS L.	2/6 ALMEYDA N.	3/6 SALAZAR J.A.	4/6 LOCK S.	5/6 MORALES A.	6/6 VILLAFAN C.
FIRMADO ROJAS L. 03/02/2010 11:15 am	FIRMADO ALMEYDA N. 03/02/2010 02:19 pm	FIRMADO SALAZAR J.A. 03/02/2010 02:25 pm	FIRMADO LOCK S. 03/02/2010 02:36 pm	FIRMADO MORALES A. 04/02/2010 10:10 am	FIRMADO VILLAFAN C. 04/02/2010 12:55 pm

26/02/2010 08:38 Av. Atocongo 2440 Villa María del Triunfo, Lima 35, Perú, Teléfono : (511) - 217-0200 Fax : 217-1497 Solo de Uso Interno \*\*

### **Is it in stock?**

Next, warehouse staff receives the product purchase request and checks the stock to see if the product is in stock. The process now subdivides:

#### **Not in stock**

If the desired item/product is not in stock, the request is sent to the Purchases department. They call up suppliers to get estimates for cost. Once they find the cheapest options, they file an "*Orden de Compra*" or purchase order (Fig. 1.2). This document includes relevant information of the supplier as well as full information of the requested product.

The item will arrive with a document called a "*Guía de Remisión*", a proof of reception (Fig. 1.3). Warehouse staff checks the item against what was requested in the purchase order. If it fits the description of what was ordered, it is admitted into the warehouse and is entered into the system. This way, the stock is kept up-to-date. A document called "*Recepción de Material*" is created, a record that the item has arrived and has been entered to the warehouse (Fig. 1.4). If it does not fit the description or is malfunctioning, it is returned.

#### **Item in Stock/Item Pickup**

The following applies to both when the item requested is in stock or when it is not and has had to be ordered, as detailed above.

The user that requested the item is informed via email that the item has arrived. He or she fills a "*Vale de salida de materiales*" or Item Exit Voucher (Fig. 1.5), and visits the warehouse to take out the items. The user hands this form to Warehouse staff.

Warehouse staff visits the aisles of the warehouse, picking each one of the requested items. The user is given a document called the "*Vale de salida de materiales*" or Item Exit Form that has information such as the product name and quantity, and is signed by the receiver and by warehouse staff.

### Fig 1.2 – Purchase Order

Includes supplier information such as name, address, telephone numbers, fax, RUC (business identifier code). Includes item-specific information like product name, quantity, description and price.

 <b>Cementos Lima S.A.</b>																																																
RUC: 20100137390 Av. Atocongo No. 2440 Telef.: 51 1 2170200 Fax: 51 1 2171497 Lima 35 - PERU																																																
<b>Pedido Local</b>																																																
Nro.: <b>20100166 - 2</b> Fecha: <b>04-Feb-2010</b>																																																
PEDIDO A: <b>IBEROAMERICANA DE SISTEMAS S.A.C.</b> RUC: <b>20418054477</b>																																																
DIRECCION: <b>AYACUCHO 172 OF. 605- CERCADO - LIMA - LIMA</b> LIMA 01																																																
TELEFONO: <b>4259903 /3452075</b> ADI: <b>8125101</b> Hardware																																																
FAX: <b>4263929</b> RESP.: <b>SBN</b>																																																
OBSERVACIONES: DESTINO: ITEM 1.- GG. ITEMS 2 Y 3.-SIS. OBS.- SE ADJUNTA COTIZACION DE OMNIA SOLUTION Y DE EMPORIUM IBEROAMERICANA DE SISTEMAS.																																																
CONDICIONES DE PAGO: <b>PRESENTACION FACTURA</b> MONEDA: <b>USD</b>																																																
<table border="1"> <thead> <tr> <th>ITEM</th> <th>CANTIDAD</th> <th>UND</th> <th>CODIGO</th> <th>DESCRIPCION</th> <th>Valor Vta. Unitario</th> <th>Descuento %</th> <th>Monto Descuento</th> <th>I.G.V.</th> <th>PRECIO VENTA</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>25.000</td> <td>U</td> <td>234101</td> <td>CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A</td> <td>62.610</td> <td>0.00%</td> <td>0.00</td> <td>19.00%</td> <td>1,862.65</td> </tr> <tr> <td>3</td> <td>5.000</td> <td>U</td> <td>223498</td> <td>MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL</td> <td>13.000</td> <td>0.00%</td> <td>0.00</td> <td>19.00%</td> <td>77.35</td> </tr> <tr> <td colspan="8"></td> <td>Total Pedido:</td> <td>USD <b>1,940.00</b></td> </tr> </tbody> </table>									ITEM	CANTIDAD	UND	CODIGO	DESCRIPCION	Valor Vta. Unitario	Descuento %	Monto Descuento	I.G.V.	PRECIO VENTA	2	25.000	U	234101	CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A	62.610	0.00%	0.00	19.00%	1,862.65	3	5.000	U	223498	MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL	13.000	0.00%	0.00	19.00%	77.35									Total Pedido:	USD <b>1,940.00</b>
ITEM	CANTIDAD	UND	CODIGO	DESCRIPCION	Valor Vta. Unitario	Descuento %	Monto Descuento	I.G.V.	PRECIO VENTA																																							
2	25.000	U	234101	CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A	62.610	0.00%	0.00	19.00%	1,862.65																																							
3	5.000	U	223498	MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL	13.000	0.00%	0.00	19.00%	77.35																																							
								Total Pedido:	USD <b>1,940.00</b>																																							
<table border="1"> <thead> <tr> <th>ITEM</th> <th>FECHA MAX. DE ENTREGA</th> <th>FORMA DE ENTREGA</th> <th>LUGAR</th> <th>ADI</th> </tr> </thead> <tbody> <tr> <td>2-3</td> <td>10/02/2010</td> <td>INMEDIATA</td> <td>ATOCONGO</td> <td></td> </tr> </tbody> </table>		ITEM	FECHA MAX. DE ENTREGA	FORMA DE ENTREGA	LUGAR	ADI	2-3	10/02/2010	INMEDIATA	ATOCONGO																																						
ITEM	FECHA MAX. DE ENTREGA	FORMA DE ENTREGA	LUGAR	ADI																																												
2-3	10/02/2010	INMEDIATA	ATOCONGO																																													
<b>DESTINO: SISTEMAS</b> <b>USUARIO</b> <small>El Tipo de Cambio a Aplicar sera el de Venta Promedio Fondo de Publicado en el Diario Oficial EL PERUANO. Efectivo en la Fecha de los Pagos. R.CAMB. 029-90-EP/90. El proveedor declarara que si algun item tiene el est. 00000000 se refiere un equipo que un comprador no puede usar directamente, es decir que el comprador debe adquirirlo y operarlo directamente con su proveedor. De lo contrario, la compra se considera que se realizo con la fecha de la factura con el año a 4 dígitos. En caso esta declaracion no resulte cierto, el proveedor se obliga a pagar una multa de 1000 soles. De lo contrario, nuestra empresa se reserva el derecho de iniciar acciones legales por concepto de daños y perjuicios, lucro cesante y daño ulterior.</small>																																																
<small>1º.- Las Facturas en dos ejemplares (ORIGINAL y COPIA S/UNAT), deben enviarse a la Gerencia inmediatamente después de ejecutarse la entrega de la mercadería, adjuntando este pedido en original. Toda demora en el pago imputable a esta omisión no es de responsabilidad para la compañía.            2º.- En la guía debe mencionarse el número del pedido, cantidad, peso y precio unitario del material.            3º.- Es indispensable que las guías que acompañan a la mercadería se extiendan en triplicado a fin de retener dos copias la Fábrica.            4º.- La recepción de mercadería en la fábrica se hará de lunes a viernes de 8 a 12 m. y de 1 a 4 p.m. Los Sábados de 8 a 12 m.            5º.- Ver hoja adjunta de COMPRAS CON LA GERENCIA.            6º.- La recepción de facturas es de Lunes a Viernes de 8 a.m. a 12 m.            7º.- Cada ORDEN de COMPRA generara una factura independiente.            8º.- Las facturas deberán ser emitidas cumpliendo todos los requisitos establecidos por el reglamento de comprobantes de pago.</small>																																																

### Fig. 1.3 – Proof of Reception

Document that the supplier hands to warehouse staff as proof that product has been delivered. Includes basic item information that is verified against the Purchase Order.

**OmniaSolution**

OMNIA SOLUTION S.A.C.  
Los Tulipanes 188 - Of. 304 Urb. El Polo Hunt  
Surco (Lima 33) - Perú  
Central: (511)437-4717 Telefax: (511)437-4679  
E-mail: postmast@omniasolution.com

**R.U.C. Nº 20475835**  
**GUIA DE REMISIC**  
**REMITENTE**  
**001 - Nº 0000615**

Lima, 8 de Febrero del 2010  
Señor(es): Cementos Lima s.a.  
Dirección: Av. Atocongo 2440 Villa María del Triunfo  
R.U.C. 20100137390

MOTIVO DEL TRASLADO  
 1. Venta  5. Devolución   
 2. Traslado de bienes para transformación  6. Traslado entre establecimientos de la misma empresa   
 3. Consignación  7. Traslado por emisor itinerante de comprobantes de pago   
 4. Compra  8. Otros

CANT.	UNIDAD	DESCRIPCION
01	Und.	Lectora DVD SAMSUNG N/S: R0546GF59606384

P. de Partida: Tulipanes 188 -  
Nº de Factura: 001-003429  
P. de Llegada: Atocongo 2440  
Fecha de Inicio del Traslado: 08-02-2

Transportista:  
R.U.C.:  
Domicilio:  
Nº Licencia:  
Marca y Nº de Placa:

CEMENTOS LIMA S.A.  
11 FEB. 2010  
CLIENTE  
SISTEMAS

RECIBI CONFORME  
DESTINA

PL - 20100166 -

YCHINFORMAS S.A. R.U.C. 2025940295  
TEL/FAX: 265-7188  
Nº 0221726021 FI: 16-07-08  
EL 091-0451 AL 001-1450

### Fig 1.4 – Item Reception Form

Form generated to enter product into stock. Includes the code of the Proof of Reception form. Includes product information such as name, state, amount, measuring unit (units, gallons, boxes, etc.).

#### Recepción de Material

No.:	<b>072049</b>	Kardex:	000082 - 201002						
Fecha :	09/02/2010	No.Guia:	<b>001-000000003999</b>						
No. Documento :	PL : 20100166.2	Proveedor:	20418054477 IBEROAMERICANA DE SISTEMAS S.A.C.						
Solicitante :	0091 0091								
Area Destino :	1220 SISTEMAS								
Observaciones:	Item 1 : Backup, ítem 2 : usuarios varios, Vale de salida No. 407119								
Itm.	Material	Activo	Descripción	Estado	Cant.	U.M.	Cant.	U.M.	ADI
1	234101		CINTA DATOS REESCRIBIBLE HP ULTRIUM DE 400 A 800 GB,160MB/s CODIGO,C7973A	Nuevo	25.00	U	25.00	U	
2	223498		MOUSES OPTICOS USB MARCA:MICROSOFT CON SCROLL	Nuevo	5.00	U	5.00	U	

### Fig 1.5 – Item Exit Voucher

Includes information such as item amount, unit of measurement, item description. It is signed by user taking out the item and warehouse staff giving the items.

Cementos Lima S.A.		VALE DE SALIDA DE MATERIALES			Nº 407119	
AREA:	1220	FECHA: 08 / 02 / 2010		REF. _____		
CODIGO	CANT.	U.M.	DESCRIPCION	C. C/TA.	ANEXO/OS. OT/ADI	COD. DE MAQUINA
	25	c/u	CINTA DATOS RECIBIDAS HP ULTRIUM CX 400 GB/SP GR	91504	OF 11021	
	5	c/u	MOUSE OPTICO USA MARCA MICROSOFT CON SCROLL	91504	OF 11021	
	7		PL 20100166-2	7		
DESTINO	Term 1. - Para Backups		RECIBIDO	DESPACHADO	AUTORIZADO	
			<i>[Signature]</i>		COD 0091 <i>[Signature]</i>	
	Term 2. - Usuarios Varios					

Cementos Lima S.A.		VALE DE SALIDA DE MATERIALES			Nº 407122	
AREA:	1220	FECHA: 11 / 02 / 2010		REF. _____		
CODIGO	CANT.	U.M.	DESCRIPCION	C. C/TA. ADT	ANEXO/OS. OT/ADI	COD. DE MAQUINA
	01	c/u	LECTOR DVD SAMSUNG NIB: R054GG + 5910-324	812	5101	
DESTINO	G6.		RECIBIDO	DESPACHADO	AUTORIZADO	
			<i>[Signature]</i>		COD 0091 <i>[Signature]</i>	

## THE CURRENT SYSTEM

As it can be expected from a large company with a stock so valuable, a proper, full-range system exists to deal with the tasks and processes detailed in the flowchart.

## Current Warehouse Operation System

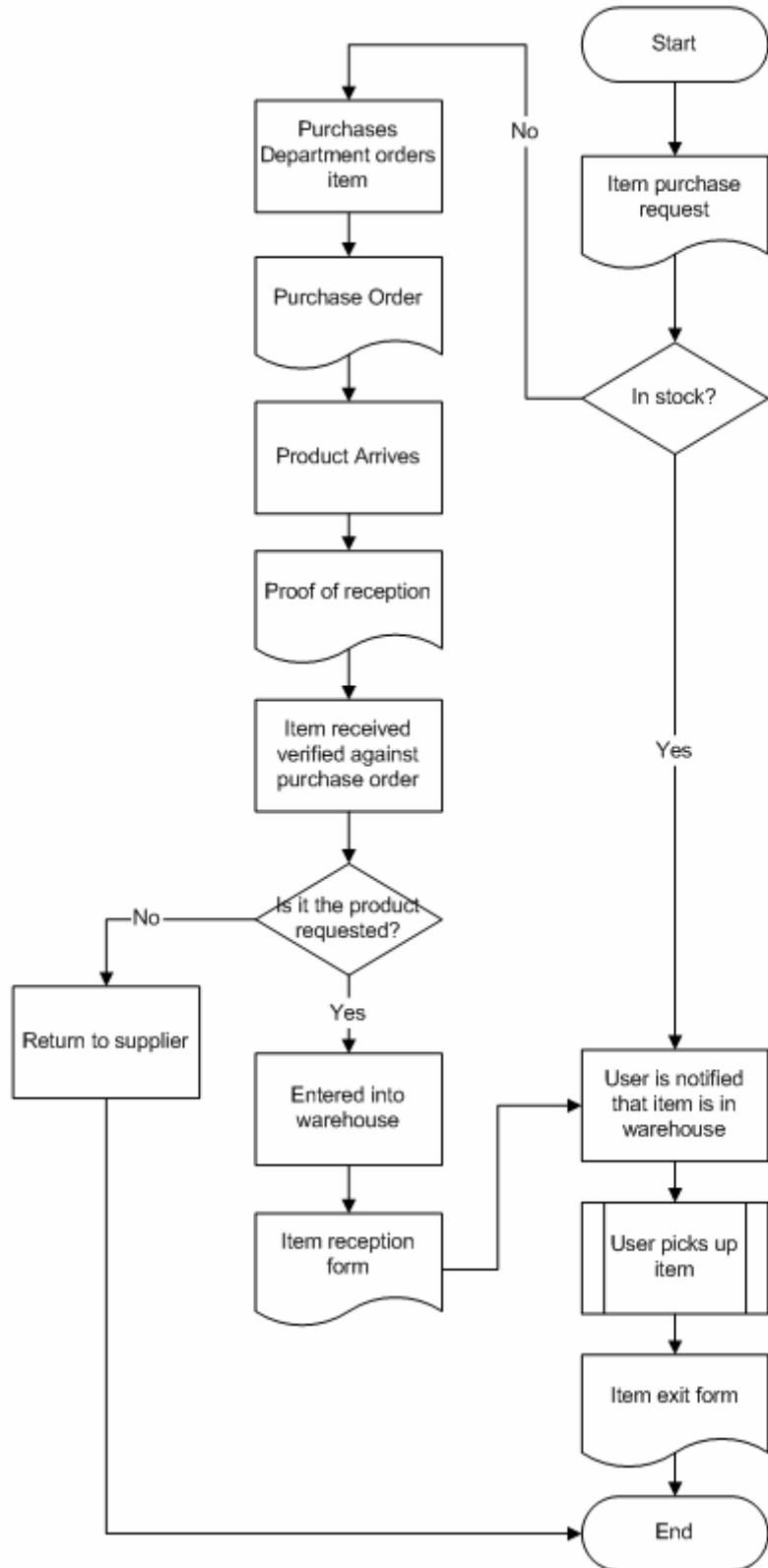


Fig. 1.6 - Warehouse Operation Flowchart

## Item Pickup Process

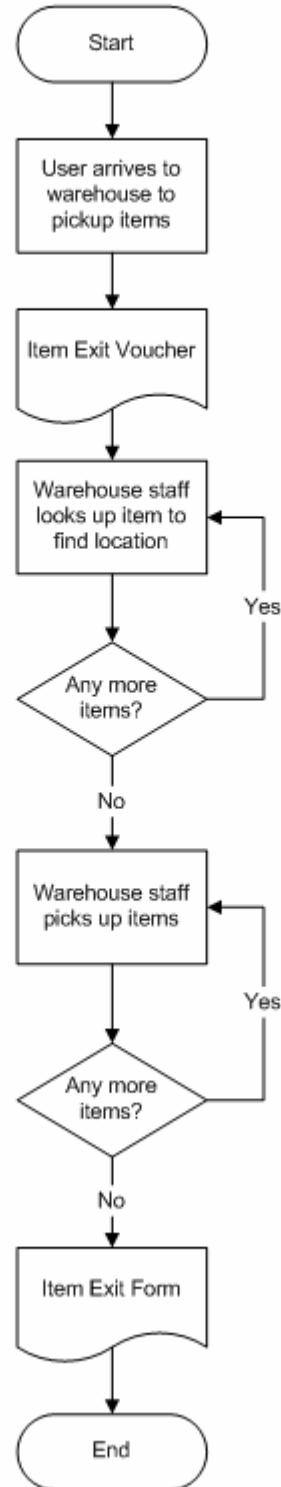


Fig. 1.7 - Item Pickup Process

## The Current Warehouse Application

Below are some screenshots of the existing application in place that deals with warehouse operation and maintenance.

### Fig 1.8 – Search screen

Items can be looked up by a variety of search criteria, including but not limited to Name, Item Group, Code, Location, Current Quantity, Status, etc. These criteria are all relevant to warehouse operation, but not necessarily relevant to those dealing with item entry or exit transactions. The search results are displayed in a table below the criteria.

The screenshot shows a Windows application window titled 'Cementos Lima S.A. - Consulta de Inventarios - 2010 - Cementos Lima S.A.'. The menu bar includes Archivo, Edicion, Herramienta, Ventana, Ayuda. The toolbar has icons for search, print, and file operations. A tab bar at the top has 'Buscar Generales', 'Otros Datos', 'Kardex', 'Compras', 'Consumos', 'Ing. vs Sal.', and 'Gráficas...'. The main area has search criteria fields: Descripción (PERNO), Actividad (ACTIVO), Grupos, Familias, Código (014303), Creado, Plano, Parte, Estado, Ubicación, and various checkboxes for permisos and criticality. Below these are filters for Número, Cantidad Actual (Todos), and Saldo Actual ME (Todos). The results grid shows items from the database, including columns for Almacén, Descripción, Código Material, Estado Material, Ubicación, Unidad Medida, Cantidad Actual, Saldo Actual ME, Costo Promedio ME, Saldo Actual MN, and Costo Promedio MN. One row is highlighted in yellow, corresponding to the search term 'PERNO'. The bottom status bar shows '776 Registro(s) Encontrado(s)'.

### Fig. 1.9 – Item Description

Once an item has been found, a tab opens to display the recorded information.

The screenshot shows the same application window with a different tab selected. The title bar is '014303 - PERO (BULON - POS 6) EN ACERO 25/20, PARA BOCA DE DESCARGA DEL HORNO I. PLANO 3691'. The left panel contains item details: Código: 014303, Código Antiguo: 240.0223.004, Unidad Medida: U, and Unidad: UNIDAD. The right panel shows the 'Situación' (Status) as 'ACTIVO' in a green box. It includes fields for Creado (11/12/1994 00:00:00), Por (empty), and Modificado por (empty). A large list of checkboxes for item properties like Parada Planta, Perceble, Crítico, etc., is shown, with 'Crítico' checked. At the bottom, there's a 'Ver Foto' button and a notes section with 'MQ 04.04.01.HR.01...' and 'HORNO ROTATIVO I'.

### Fig. 1.10 – Item Transaction Records

Each item has a record of the entry or exit transactions, each with its own date. The user may choose to display transactions only between a specified time period. The current balance (quantity of the item in stock) is also displayed.

## PROBLEMS AND SHORTCOMINGS

I initially set out to improve the warehouse tracking and management system at place in Cementos Lima, but when I met with Sr. Lock and saw and used the program, I was quickly dissuaded. The system deals with several tasks very efficiently, quickly, and in a specific manner to the business. It links and merges into other aspects of the company, like Accounting, Finances, etc.

It would be redundant to develop aspects such as Billing, rotation cycles for stock or accurate projections of supply and demand. Time constraints would not allow for a program so complete, plus most of the tasks have no need to be improved as with the current stock it is working just fine.

In an interview with Sr Jaime Salazar, the manager of the warehouses at CL, he repetitively stressed the need for efficiency. "We are the backbone of the business; we have to ensure that we deliver in the shortest time possible."

While touring the four warehouses, I observed that most of the activity was centred on Warehouse A. It stores smaller products, ranging from tools to nails and bolts. It holds several thousand products, and most can be taken out

manually. The perimeter of the warehouse does hold some heavy items that need to be picked up with machines.

As I was taking the tour of the warehouse, I happened to witness a person coming in to take out a few items. Warehouse staff followed the process detailed above: the manually looking for products and picking them up, and the item exit form being filled.

I noticed a possible time drain in that process. If a user needs, say, 15 items, from bolts to extensions to safety gloves, warehouse staff may very well spend quite some time collecting the products. The system in place does give the exact location of an item in the warehouse, but warehouse staff rarely use it.

A few reasons for this might be:

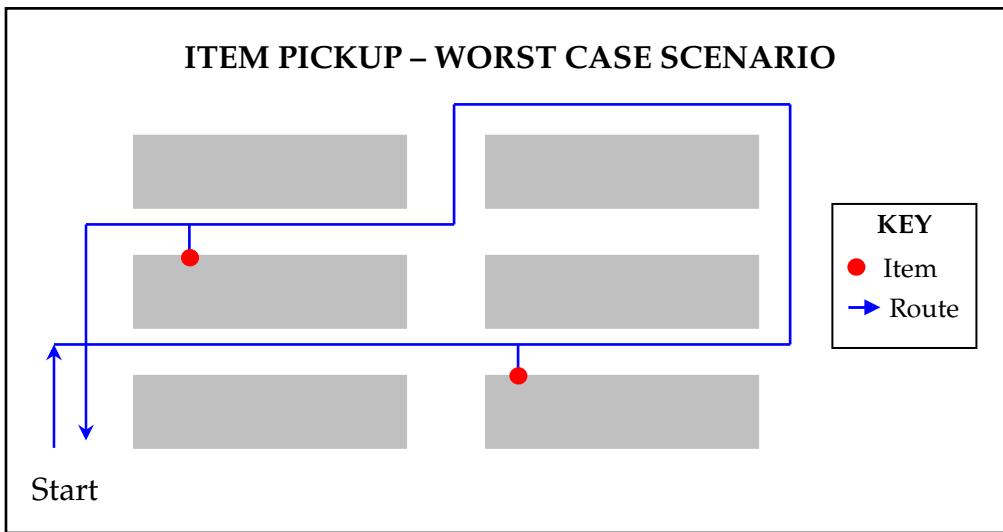
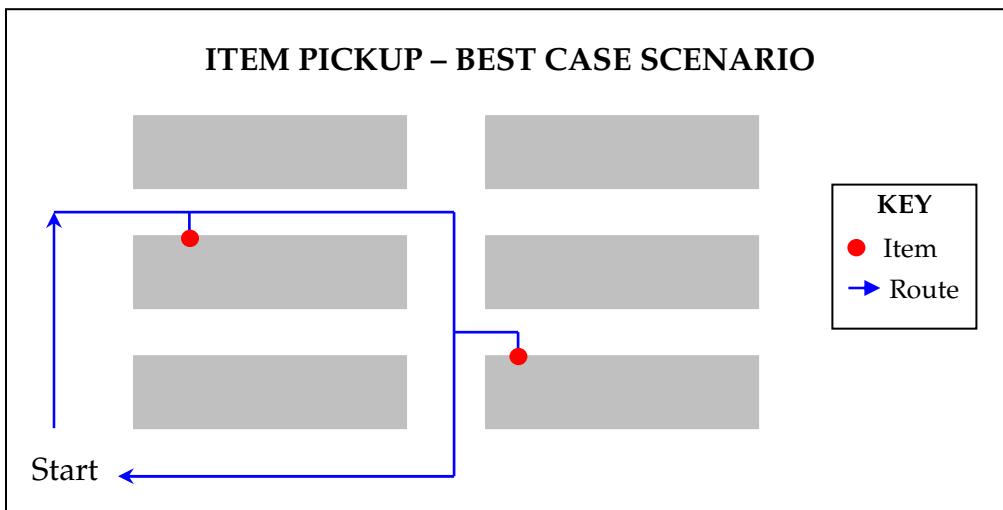
- Experience has made them vaguely know the location of products
- The system looks up items individually: looking for  $n$  products over and over again might take a long while
- The search function does not relate specifically to the task of collecting products, it is just a general “google search”.

This was corroborated by asking one of the warehouse workers on duty at the time of my visit. Workers either know most of the locations of the items in the warehouse. The times when they don't, they normally just scribble the product location in the back of their hand.

Most probably an experienced warehouse worker will not get lost looking for a product, and will at least have a general idea of the item location. However, what could end up happening is that while looking for those  $n$  products they might retrace their steps or use an inefficient route.

As a side note, the user stressed the importance of having users with security measures like passwords and special privileges in any warehouse application, to avoid unauthorized personnel changing the highly-important stock information.

Fig. 1.11 – Best/Worst Case Item Pickup example with 2 items



## LAYOUT OF WAREHOUSE 1

This storage facility holds assorted products, while other facilities focus in few types of products (e.g. warehouse 2 only stores cement bags, warehouse 3 only stores large machinery).

The warehouse has two distinct areas, the outer and inner ones. The outer racks hold large items or machines that cannot be withdrawn without the help of a machine or multiple individuals. The inner area has 70 aisles of racks, denoted by numbers 1-70, each with 5 rows denoted by letters A-E and 10 columns denoted by numbers 1-10.

The diagram below illustrates a rough outline of the warehouse. The green dot shows the position of the desk where item withdrawal requests are attended.

1	37
2	38
3	39
4	40
5	41
6	42
7	43
8	8
9	44
10	45
11	46
12	47
13	48
14	49
15	50
16	51
17	52
18	53
19	54
20	55
21	56
22	57
23	58
24	59
25	60
26	61
27	62
28	63
29	64
30	65
31	66
32	67
33	68
34	
35	69
36	70



Fig. 1.12 – Warehouse layout

The existing system stores the location of each item in a standard format. This format is

ANN.AA.RCC

Where 'A' is an abbreviation for Almacén (warehouse in spanish)

NN – warehouse number

AA – aisle number

R – row number

CC – column number

The diagram below illustrates an example location.

**Fig 1.13 – Example location**

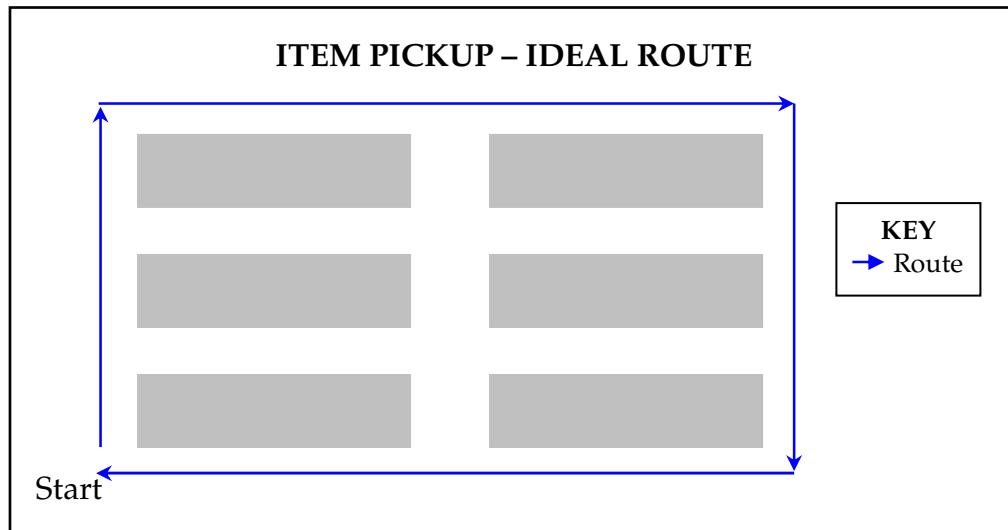
Aisle 38 in Warehouse 1										
	1	2	3	4	5	6	7	8	9	10
A										
B										
C										
D										
E										

The highlighted location is  
**A01.38.C03**

A warehouse worker processing an item withdrawal request will make his way from the desk to the aisles, pickup each process and return to the desk.

Speaking with some warehouse workers, they agreed that the best route was always to start from the lower leftmost aisles, and make their way around the warehouse until they return to the starting point. This ideal route is illustrated below.

Fig 1.14 – Ideal route



\* \* \*

The shortcomings of the system that currently handles warehouse management and operation can be summarized as follows:

Lack of a proper, specialised way of handling item entry and exit operations that allows for multiple product searches and calculates an optimal route for item pickup.

## A2 – Criteria for Success

The computer-based solution for the item entry and pickup processes should be able to perform the following functions:

Objective	Description
1	Item Records
1.1	Creating an item
1.1.1	- User must be able to create a new item
1.1.2	- Item must be assigned an ID (a unique integer identifier) by the system
1.1.2	- Item must be assigned a creation date. The date should not be before the current date.
1.1.3	- The username of the warehouse worker that created the item must be assigned to it
1.1.4	- User must be able to input the following information with proper validation. Fields marked with an * are required <ul style="list-style-type: none"><li>○ Name* – text and/or numbers, no more than 200 characters</li><li>○ Code – a 6 digit code, the code assigned to the item in the existing warehouse application</li><li>○ Group* – the group that the item belongs to</li><li>○ Unit of measurement* – the unit by which the item is measured (e.g. unit, litre, gallon, etc.)</li><li>○ Description – text and/or numbers, no more than 200 characters</li><li>○ Image</li><li>○ Location – the location of the item in the warehouse, of type A01.AA.RCC where A01 denotes that the item is inside warehouse 1, AA is the aisle number, R is the row and CC is the column number</li></ul>
<i>Note: all the validation rules explicitly requested by the user</i>	
1.2	Updating item information
1.2.1	- User must be able to update item information
1.2.2	- User should be able to update all the fields listed in objective 1.1.3. These updates must be constricted to the validation rules detailed in that objective.
1.2.3	- User must be able to permanently delete an item
1.3	Looking up an item
1.3.1	- User must be able to look up an item individually by one or more of the following criteria: <ul style="list-style-type: none"><li>○ Code</li></ul>

	<ul style="list-style-type: none"> <li><input type="radio"/> Name</li> <li><input type="radio"/> Group</li> <li><input type="radio"/> Location</li> </ul>
1.3.2	<ul style="list-style-type: none"> <li>- User must be able to click on preliminary search results to view full item information</li> </ul>
2	Stock Maintenance
2.1	<ul style="list-style-type: none"> <li>- User must be given the option to register the arrival of an item. User must be able to add the amount of the item that has arrived based on the unit of measurement.</li> </ul>
2.2	<ul style="list-style-type: none"> <li>- User must be able to register the exit of an item. This will be solved through Objective 3.5</li> </ul>
2.3	<ul style="list-style-type: none"> <li>- A record of item entries/exits must be kept, showing the date and amount. Date must not be before the current date.</li> </ul>
3	Item Withdrawal
3.1	<ul style="list-style-type: none"> <li>- User must be able to search for multiple products by one or more of the following criteria:           <ul style="list-style-type: none"> <li><input type="radio"/> Code</li> <li><input type="radio"/> Name</li> <li><input type="radio"/> Group</li> <li><input type="radio"/> Location</li> </ul> </li> </ul> <p>They must be added to a list of items to process.</p>
3.2	<ul style="list-style-type: none"> <li>- User must be given the option to input the integer quantity of the item to be withdrawn. It must not be greater than the amount in stock; else an error has to be thrown.</li> </ul>
3.3	<ul style="list-style-type: none"> <li>- User must be given the option to calculate the best route to pickup the items. This will be done based on item locations. A clear order of items in the pickup route must be shown.</li> </ul>
3.4	<ul style="list-style-type: none"> <li>- When route is calculated, the system has to assume that products have been taken out, and the stock has to be updated by registering item exits.</li> </ul>
4	Usability
4.1	<ul style="list-style-type: none"> <li>- The program will be used by warehouse staff, low-experienced computer users. The program has to be designed with that intended user group, and must be simple and straightforward.</li> </ul>
4.2	<ul style="list-style-type: none"> <li>- The application must have a user-friendly Graphical User Interface</li> </ul>
4.3	<ul style="list-style-type: none"> <li>- The application must have a menu bar where different functions can be accessed.</li> </ul>
4.4	<ul style="list-style-type: none"> <li>- Different functions should display in the main screen as buttons, clearly stating their function.</li> </ul>
4.5	<ul style="list-style-type: none"> <li>- The user has to know exactly what to enter. Fields should clearly describe their function; required fields</li> </ul>

	must be distinctly marked.
5	User Management
5.1	Regular user
5.1.1	<ul style="list-style-type: none"> <li>- The user must be able to log into the system using a given username and password</li> </ul>
5.1.2	<ul style="list-style-type: none"> <li>- The user must be able to access all functions related to warehouse maintenance and operation detailed in the objectives above.</li> </ul>
5.1.3	<ul style="list-style-type: none"> <li>- The user must be able to change his/her password</li> </ul>
5.2	Admin
5.2.1	<ul style="list-style-type: none"> <li>- The system admin must be able to log into the system using a given username and password</li> </ul>
5.2.2	<ul style="list-style-type: none"> <li>- A user with administrator permissions must be able to access all functions related to warehouse maintenance and operation detailed in the objectives above.</li> </ul>
5.2.3	<ul style="list-style-type: none"> <li>- Admin must be able to create a user with the following information           <ul style="list-style-type: none"> <li>o Username – no more than 20 characters or numbers</li> <li>o Password – no more than 30 characters or numbers</li> <li>o User Permissions – Admin, Entry, Exit</li> <li>o User status – enabled or disabled</li> <li>o User's full name</li> </ul> </li> </ul>
5.2.4	<ul style="list-style-type: none"> <li>- Admin must be able to update user information with the fields detailed in objective 5.2.3</li> </ul>
6	Performance
6.1	<ul style="list-style-type: none"> <li>- The application must be quick and run smoothly. Efficiency is required in warehouse operation; the application has to reflect this.</li> </ul>
6.2	<ul style="list-style-type: none"> <li>- Searching and sorting must be efficient and quick</li> </ul>
6.3	<ul style="list-style-type: none"> <li>- The optimal route must be calculated quickly, with no apparent delay for the user to notice</li> </ul>
6.4	<ul style="list-style-type: none"> <li>- The application must be efficient in memory use and storage. It must not exceed the minimum requirements of the Java 6 Virtual Machine.</li> </ul>

---

## A3 – Prototype Solution

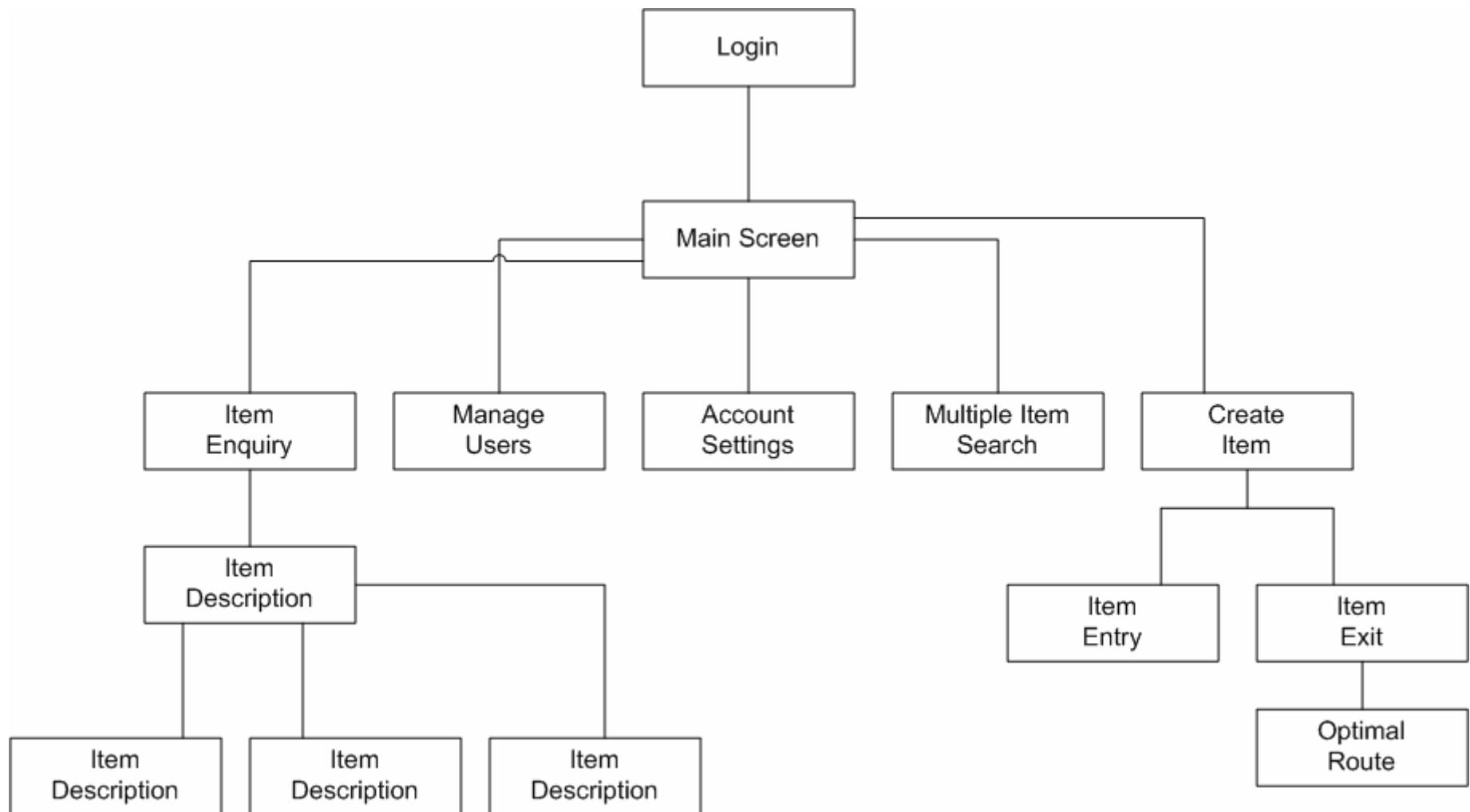
---

A GUI will be developed to satisfy the objectives in section A2. Non-functional prototypes have been developed in Java using the Jigloo GUI Builder.

The screens were shown to Sr Santiago Lock, the system admin at Cementos Lima. Having developed the Warehouse application of the CeliSistemas suite, he has the knowledge of the operations and the target users.

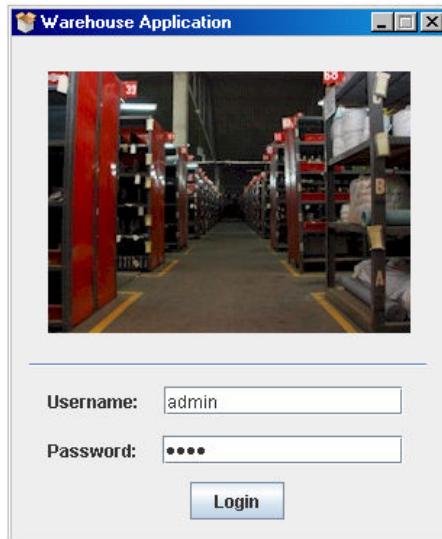
First, an initial design was produced to serve as the basis of the prototypes.

## INITIAL DESIGN



## LOGIN SCREEN

This screen will launch at start up, and will allow a user to log into the system by providing a username and password.

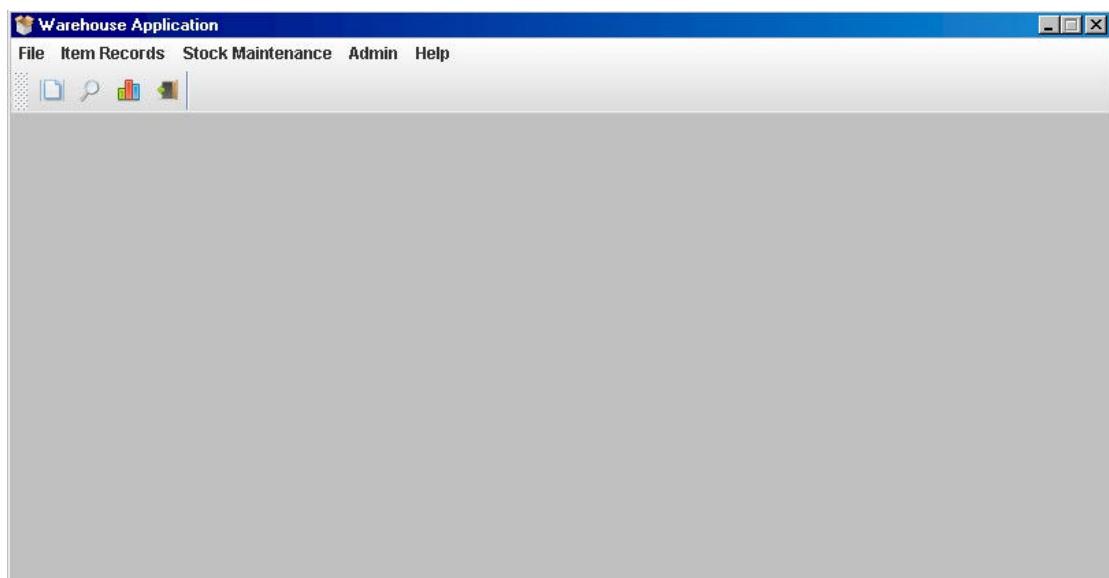


*User Comments:*

The screen was approved by the user.

## MAIN SCREEN

This will be the application's main screen, where all can be accessed.



**User Comments:** The user approved this screen.

## NAVIGATION

### Menus



#### *Specific User Comments:*

The user asked if the “Account Settings” option should be included inside the application after login or in the login screen. After some discussion and cross-reference with the login process in Windows XP, he agreed that the best alternative was to have this option inside the application.



#### *Specific User Comments:*

The user asked if this menu item would be displayed to a non-admin user. I replied that a validation check would be made on login and if the user did not have administrator permissions, the option to manage users would not be displayed. He agreed to this.



*General User Comments:*

With regards to appearance, the user approved the menus and thought them straightforward and appropriate for the target user group. Sr Lock raised a very important question. He asked about the types of users that would have access to this application. My reply, as stated in the original objective 5.2.3, was that users with administrator or standard privileges could be created.

He asked for a division of the “standard” user privileges. Sr Lock said that the entry and exit processes were handled by different people. Those that handle item entries normally have more experience with billings and accounting.

I thought that his suggestion was very sensible, and agreed to the following user functions breakdown: administrator, entry, exit. In addition, if staff were to be absent, the admin can enable extra permissions temporarily (or permanently).

This reflects on the UI. An admin user will be able to see and access all navigation options. A user with entry permissions will not be able to see the exit functions, and vice versa. The functions accessible to each type of user are:

1. Admin User

File (Account Settings)  
Item Records (Search + Create)  
Stock Maintenance (Entry + Exit)  
Admin (Manage Users)  
Help (About)

2. User with Item Entry permission

File (Account Settings)  
Item Records (Search + Create)  
Stock Maintenance (Entry)  
Help (About)

2. User with Item Exit permission

File (Account Settings)  
Item Records (Search)  
Stock Maintenance (Exit)  
Help (About)

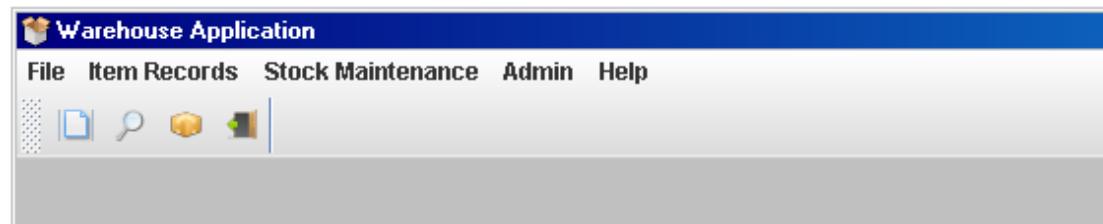
### Toolbar

A toolbar will allow a user to access main application functions. It will have four icons: Create a new Item, Item Enquiry, Stock Entry, Item Exit. It will be visible in every screen, and tooltip text for each icon detailing its function will be provided.



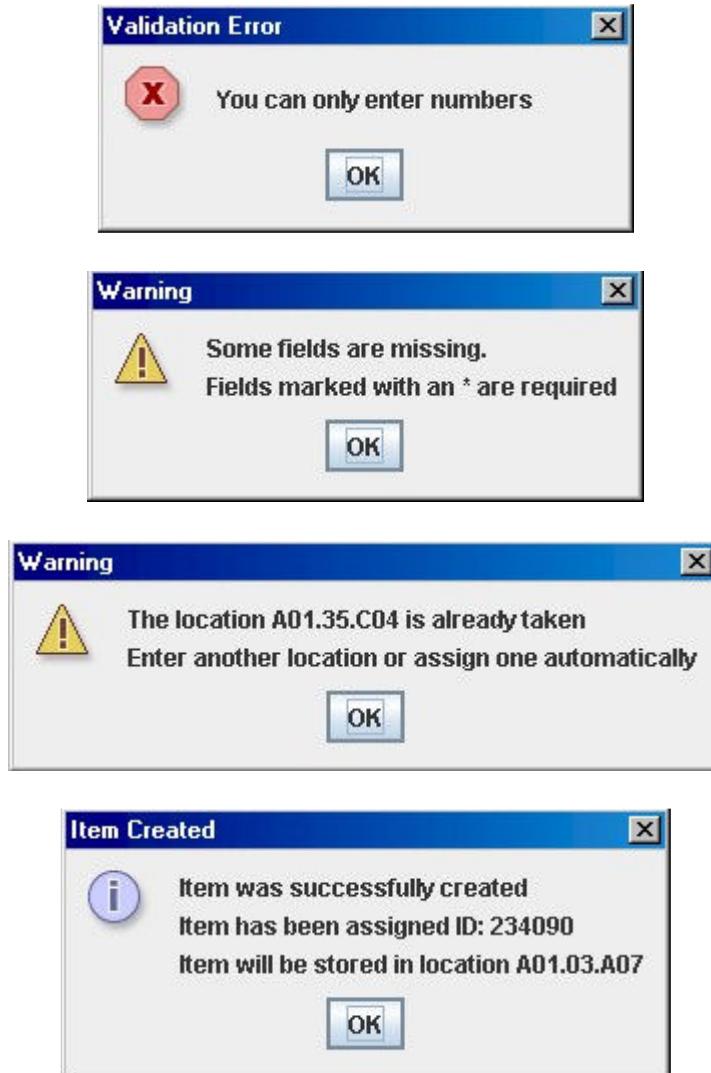
#### *User Comments:*

The user thought that the toolbar was visually appealing and had the necessary functions for good and easy accessibility to the application's functions. The only issue raised was that perhaps the icon for item entry (third one from left) did not clearly reflect its function. I agreed, and the screenshot of the updated image is shown below.



## DIALOGS

The following are screenshots of the dialogs that alert the user of errors and warnings in the application. The title of the dialogs explains their function.



### User Comments

The user approved the dialogs, but raised an important question regarding the warning when a location is taken. He talked about how a location of the type "A01.35.C04" makes reference to quite a big space. For small items, not a whole 1m<sup>2</sup> shelf is needed, so several items are assigned to one location.

The agreed solution is to have instead of a "fatal error" type warning that does not let you continue, to have an information dialog that shows the items stored in that "A01.35.C04". The user will be then asked to proceed or to select another location.

## ACCOUNT SETTINGS

This screen will allow a user to change his/her password.



*User comments:*

The user approved this screen.

## MANAGE USERS

This screen will allow an admin user to edit, modify and/or delete a user. To create a new user, the admin presses “New User” and a new line appears in the table. Changes do not take effect until the “Save” button is pressed.

The screenshot shows a window titled "Warehouse Application" with a blue header bar. The menu bar includes "File", "Item Records", "Stock Maintenance", "Admin", and "Help". Below the menu is a toolbar with icons for file operations. The main area is titled "Manage Users" and contains a message: "Changes will not take effect until 'Save' is pressed". A table lists users with columns: Username, Password, Full Name, Admin, Entry, Exit, and Enabled. The table data is as follows:

Username	Password	Full Name	Admin	Entry	Exit	Enabled
admin	root	System Admin	true	true	true	true
moralea	god	Alvaro Morales	false	true	false	true
locks	1234	Santiago Lock	false	true	false	true
[empty]						

At the bottom are three buttons: "New User", "Delete", and "Save".

*User comments:*

The user approved this screen

## CREATE A NEW ITEM

This screen allows a user to create a new item.

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

**Create a New Item**  
Fields marked with an \* are required

Name\*:

Family\*:  Sistemas

U.M.\*:  Units (U)

Description:

Created By:

Date:

Image:

Browse...

Location\*:  Automatically assign location  
 Manually assign location

Aisle\*:

Column\*:

Row\*:

**Create Item**

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

**Create a New Item**  
Fields marked with an \* are required

Name\*:  Cinta Datos Reescribible HP Ultrium

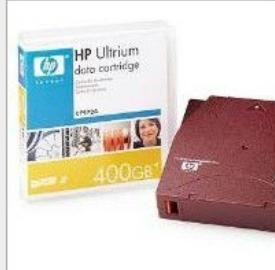
Family\*:  Sistemas

U.M.\*:  Units (U)

Description:  De 800 GB, 160 Mb/s

Created By:  MORALES A.

Date:  02/04/10

Image: 

Browse...

Location\*:  Automatically assign location  
 Manually assign location

Aisle\*:

Column\*:

Row\*:

**Create Item**

*User Comments:*

The user liked the interface to create a new item. He had a few suggestions. The user asked for the “Name” text field to be grown to fit 200 characters, instead of the 100 originally stated in objective 1.1.3. This changed the objective.

He asked the “Family” field to be renamed “Item Group”. He said that an “Automatically assign location” function was not required, and I agreed to remove it.

He asked about the item image size, and I replied that it was 200x200px. He said that the current standard is 320x240px and would rather have it that way. I agreed to his request, but said that the size of the insert item field would stay the same due to space constraints, but that the actual image size in the product description popup would be the desired size.

Finally, he asked for validation rules to be explicit and easily understandable. A screenshot of the updated “Create New Item” screen is shown below.

The user also asked that the location be set by combo boxes, so that the user cannot make a mistake setting a location.

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

Create a New Item

Fields marked with an \* are required

Name\*: Cinta Datos Reescribible HP Ultrium  
No more than 200 characters

Group\*: Sistemas

U.M.\*: Units (U)

Description: De 800 GB, 160 Mb/s  
No more than 200 characters

Image:  
  
Must be 320x240 pixels

Browse...

Created By: MORALES A. Date: 02/04/10

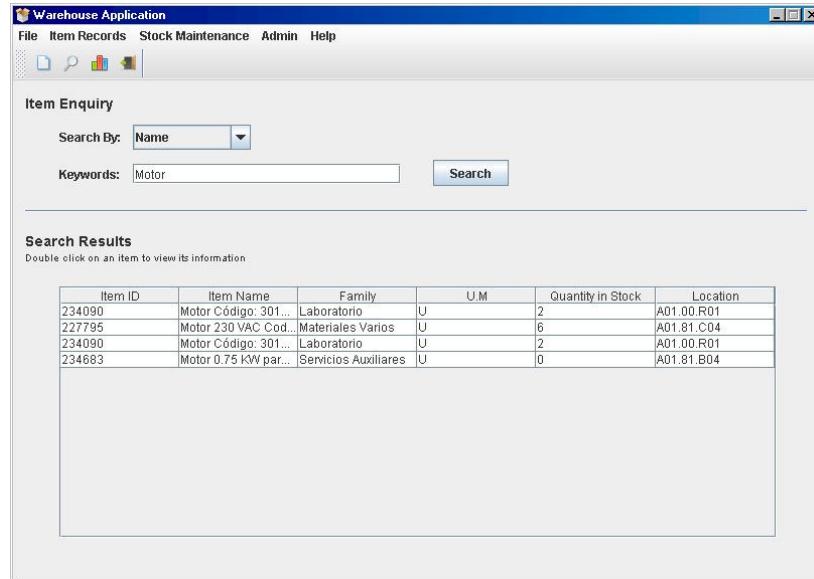
Location

Aisle\*: 36 Row\*: A Column\*: 01

Create Item

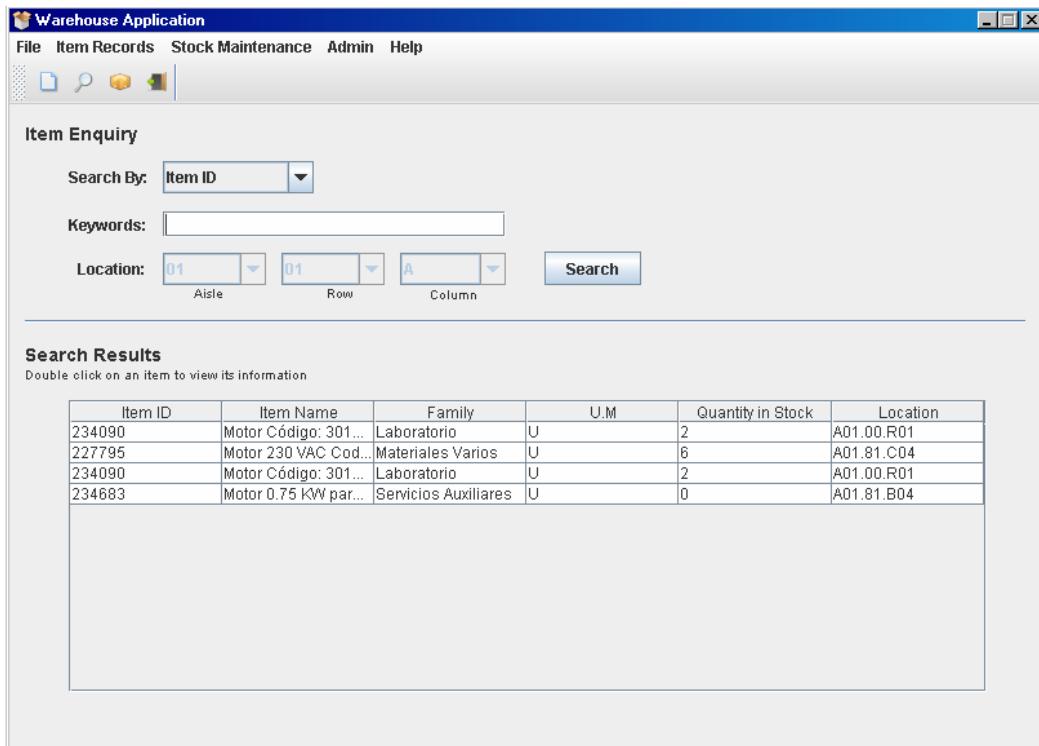
## ITEM ENQUIRY

This screen allows a user to lookup an item. The item may be searched by code, name, group or location. The search results are displayed in a table. An item may be selected and double clicked to view further information.



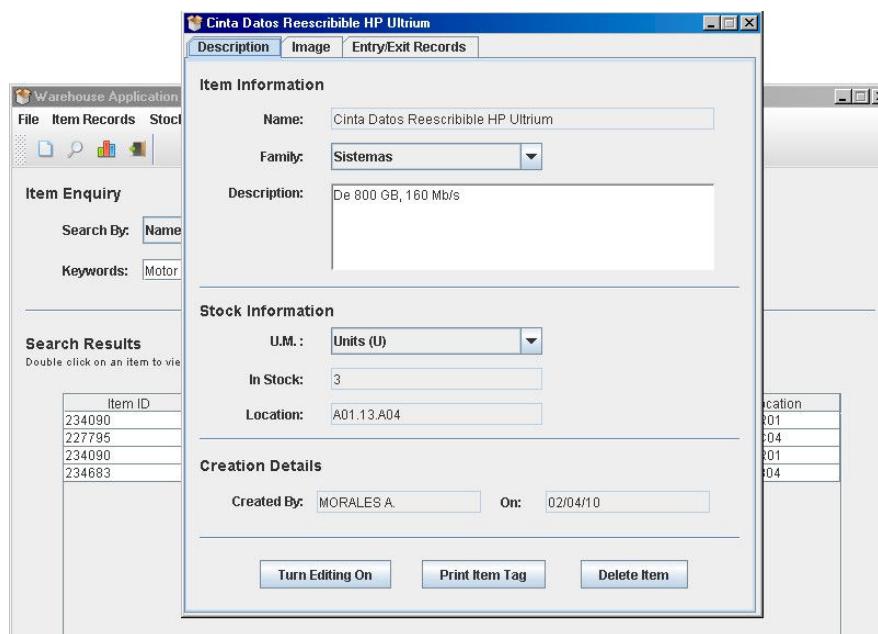
### User Comments

The user liked the search interface, and requested that location be another search criteria. This will be done by combo boxes. This changed objective 1.3.1. An updated screenshot is shown below.



## ITEM DESCRIPTION

When an item is double-clicked, the pop-up shown below is displayed for a thorough description of the item, in a tabbed interface.



**Cinta Datos Reescribible HP Ultrium**

Description    Image    Entry/Exit Records

**Item Information**

Name: Cinta Datos Reescribible HP Ultrium

Family: Sistemas

Description: De 800 GB, 160 Mb/s

**Stock Information**

U.M.: Units (U)

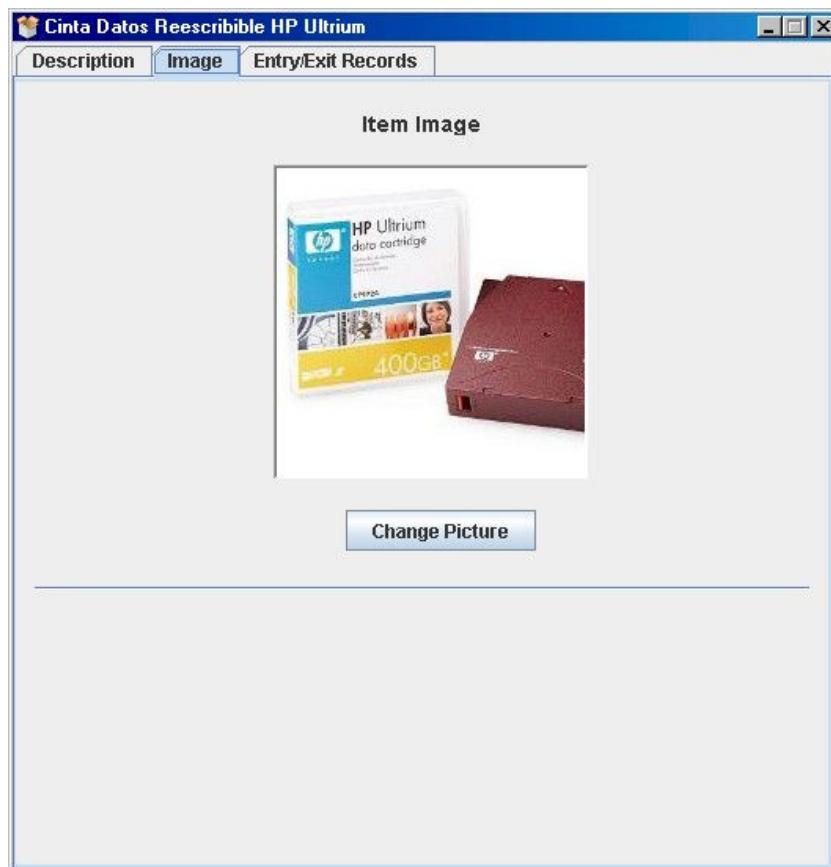
In Stock: 3

Location: A01.13.A04

**Creation Details**

Created By: MORALES A.    On: 02/04/10

**Buttons:** Save Changes    Print Item Tag    Delete Item



**Cinta Datos Reescribible HP Ultrium**

Description Image Entry/Exit Records

**Item Records**

U.M.: Units (U)

Date	Type	Quantity	Document
10/02/2010	Entry	1	003999
21/02/2010	Exit	1	407119
04/03/2010	Entry	3	041888
14/03/2010	Exit	1	071444

Total in Stock: 2

**Information**  
For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

**Cinta Datos Reescribible HP Ultrium**

Description Image Entry/Exit Records

**Item Records**

U.M.: Units (U)

Delivered By	Requested By	Warehouse Staff
Emporium S.A.C		MORALES A.
	LOCK S.	MORALES A.
Emporium S.A.C		MORALES A.
	LOCK S.	MORALES A.

Total in Stock: 2

**Information**  
For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

*User Comments:*

The user liked the item description screens and had a few suggestions.

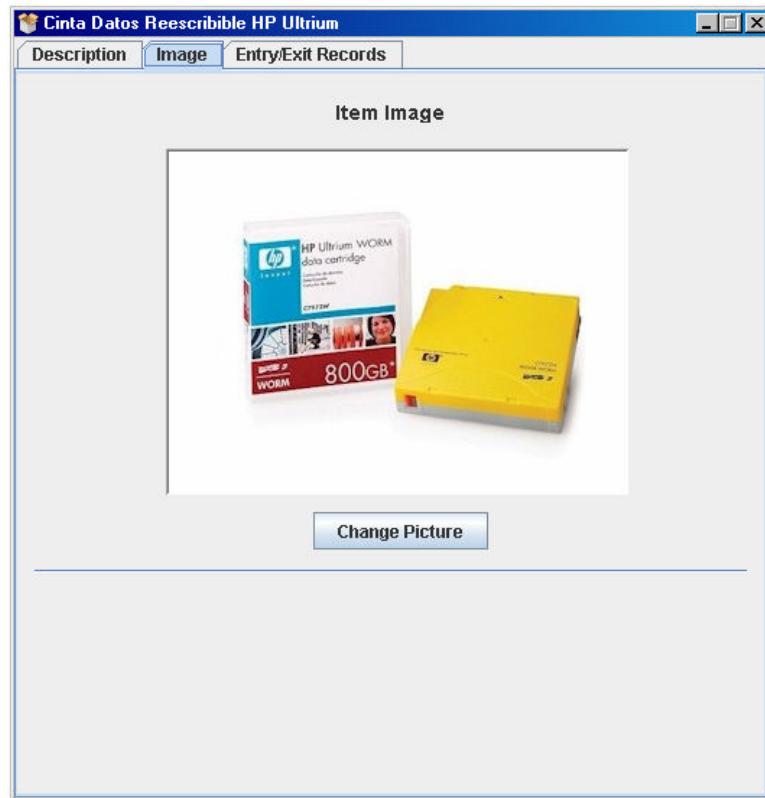
For the “Description” tab, he asked that when the “Turn Editing On” button is pressed, the field showing stock quantity should not be editable. This function should only be accessed through entry and exit processes. In addition, he asked for combo boxes to change the location as a user may enter an invalid location.

For the “Image” tab, the image size will be changed as discussed above.

For the “Entry/Exit Records” tab, the user requested several changes. First, he wanted the option to decide the time period of these records, so as to see entries and exits from a period of time you specify. If an item is 10 years old, entries and exits could border the hundreds. In addition, he wanted these entries/exits to be ordered by descending date.

He asked for a “Balance” field to be included, because a user might want to see how much stock remained after an entry or exit operation. This changes the “Total” field to “Current balance”.

Screenshots with the changes are shown below.



**Cinta Datos Reescribible HP Ultrium**

Description	Image	Entry/Exit Records																										
<b>Item Records</b>																												
From:	01	December	2000																									
Until:	01	April	2010																									
<b>Search</b>																												
U.M.: Units (U)																												
<table border="1"> <thead> <tr> <th>Date</th> <th>Type</th> <th>Quantity</th> <th>Balance</th> <th>Document</th> </tr> </thead> <tbody> <tr> <td>14/03/2010</td> <td>Entry</td> <td>1</td> <td>1</td> <td>003999</td> </tr> <tr> <td>04/03/2010</td> <td>Exit</td> <td>1</td> <td>0</td> <td>407119</td> </tr> <tr> <td>10/02/2010</td> <td>Entry</td> <td>3</td> <td>3</td> <td>041888</td> </tr> <tr> <td>06/02/2010</td> <td>Exit</td> <td>1</td> <td>2</td> <td>071444</td> </tr> </tbody> </table>				Date	Type	Quantity	Balance	Document	14/03/2010	Entry	1	1	003999	04/03/2010	Exit	1	0	407119	10/02/2010	Entry	3	3	041888	06/02/2010	Exit	1	2	071444
Date	Type	Quantity	Balance	Document																								
14/03/2010	Entry	1	1	003999																								
04/03/2010	Exit	1	0	407119																								
10/02/2010	Entry	3	3	041888																								
06/02/2010	Exit	1	2	071444																								
<input type="button" value="&lt;"/> <input type="button" value="&gt;"/>																												
Current Balance: 2																												
<b>Information</b>																												
<small>For entries, the 'Document' column gives the number of the Proof of Reception For exits, the 'Document' column gives the number of the Item Exit Voucher</small>																												

## MULTIPLE ITEM SEARCH

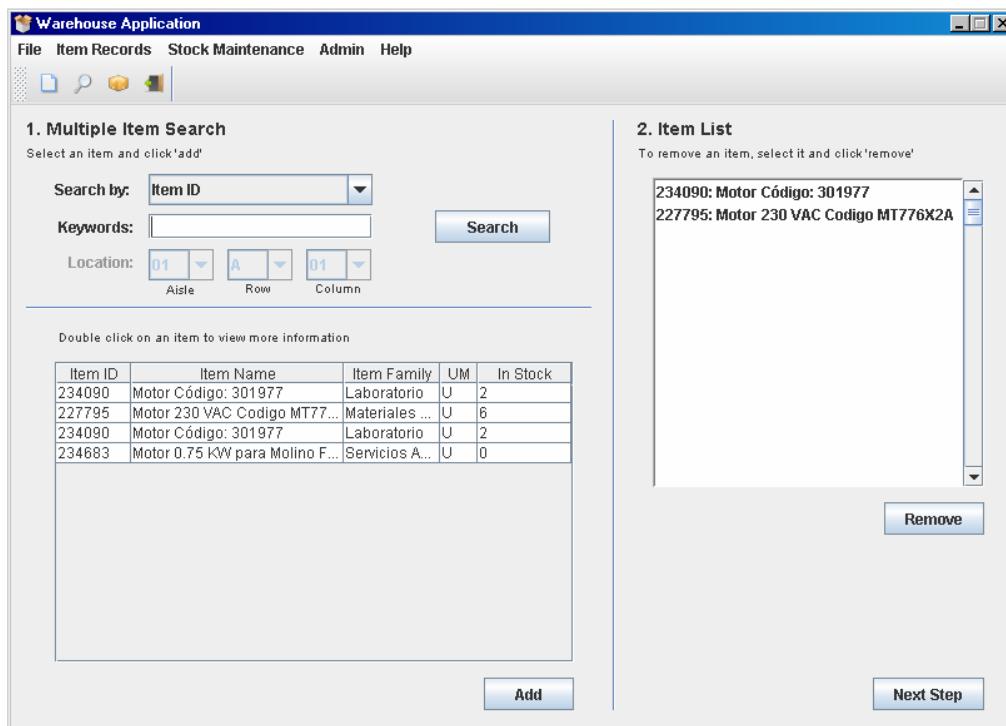
This screen will allow a worker to search for multiple items and add them to a list. These are the items that are either being withdrawn or are arriving to the warehouse.

**Warehouse Application**

File Item Records Stock Maintenance Admin Help																														
<b>1. Multiple Item Search</b> Select an item and click 'add' Search by: Name <input type="text"/> Keywords: Motor <input type="text"/> <input type="button" value="Search"/>			<b>2. Item List</b> To remove an item, select it and click 'remove' 234090: Motor Código: 301977 227795: Motor 230 VAC Código MT77... <input type="button" value="Remove"/>																											
Double click on an item to view more information <table border="1"> <thead> <tr> <th>Item ID</th> <th>Item Name</th> <th>Item Family</th> <th>UM</th> <th>In Stock</th> </tr> </thead> <tbody> <tr> <td>234090</td> <td>Motor Código: 301977</td> <td>Laboratorio</td> <td>U</td> <td>2</td> </tr> <tr> <td>227795</td> <td>Motor 230 VAC Código MT77...</td> <td>Materiales ...</td> <td>U</td> <td>6</td> </tr> <tr> <td>234090</td> <td>Motor Código: 301977</td> <td>Laboratorio</td> <td>U</td> <td>2</td> </tr> <tr> <td>234683</td> <td>Motor 0.75 KW para Molino F...</td> <td>Servicios A...</td> <td>U</td> <td>0</td> </tr> </tbody> </table>			Item ID	Item Name	Item Family	UM	In Stock	234090	Motor Código: 301977	Laboratorio	U	2	227795	Motor 230 VAC Código MT77...	Materiales ...	U	6	234090	Motor Código: 301977	Laboratorio	U	2	234683	Motor 0.75 KW para Molino F...	Servicios A...	U	0	<b>3. Select an Operation*</b> <input checked="" type="radio"/> Item Entry <input type="radio"/> Item Exit <input type="button" value="Add"/> <input type="button" value="Next Step"/>		
Item ID	Item Name	Item Family	UM	In Stock																										
234090	Motor Código: 301977	Laboratorio	U	2																										
227795	Motor 230 VAC Código MT77...	Materiales ...	U	6																										
234090	Motor Código: 301977	Laboratorio	U	2																										
234683	Motor 0.75 KW para Molino F...	Servicios A...	U	0																										

*User comments:*

The user liked the ability to search for multiple items. He requested location to be included in the search criteria. As user permissions changed and the functions became related to individual processes not both entry and exit, step 3 was removed. A screenshot of the updates is shown below.



## ITEM ENTRY

Once the items have been added to the list, this screen allows the employee to enter the quantities of each item's unit of measurement that are being entered into stock.

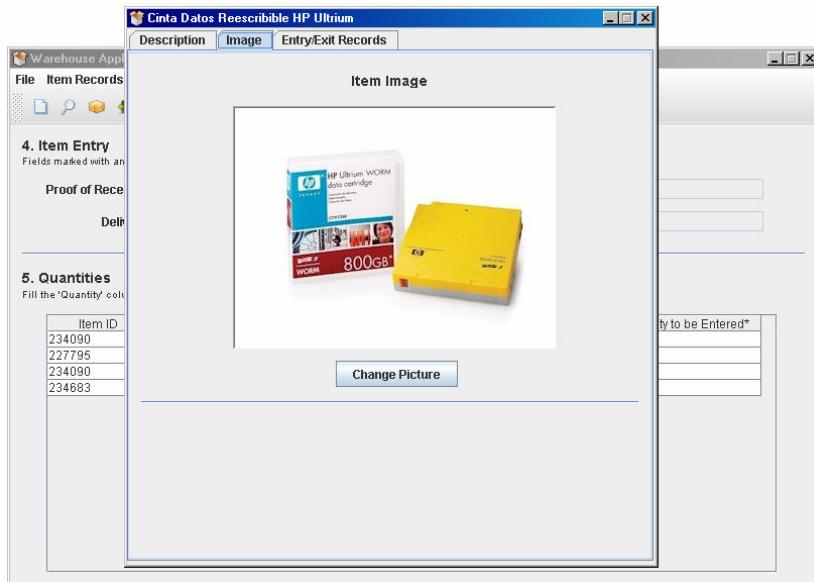
The screenshot shows the 'Warehouse Application' window with the title '4. Item Entry'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu are standard icons for file operations. The main area is titled '4. Item Entry' and contains instructions: 'Fields marked with an \* are required'. It has four input fields: 'Proof of Reception No.\*' (003999), 'Received By' (MORALES A.), 'Delivered By' (Emporium S.A.C.), and 'Date' (02/04/10). A section titled '5. Quantities' follows, with a note: 'Fill the 'Quantity' column for each item to be entered into stock'. A table lists items with their details and quantity columns:

Item ID	Name	UM	In Stock	Quantity to be Entered*
234090	Motor Código: 301977	U	2	
227795	Motor 230 VAC Código MT776X2A	U	6	
234090	Motor Código: 301977	U	2	
234683	Motor 0.75 KW para Molino Fino	U	0	

A large empty rectangular area is present below the table, likely for additional entries. A 'Finish' button is located at the bottom right of the form.

*User comments:*

The user approved the interface and arrangement. He requested that when an item is double-clicked, the item image should display.



## ITEM EXIT PROCESS

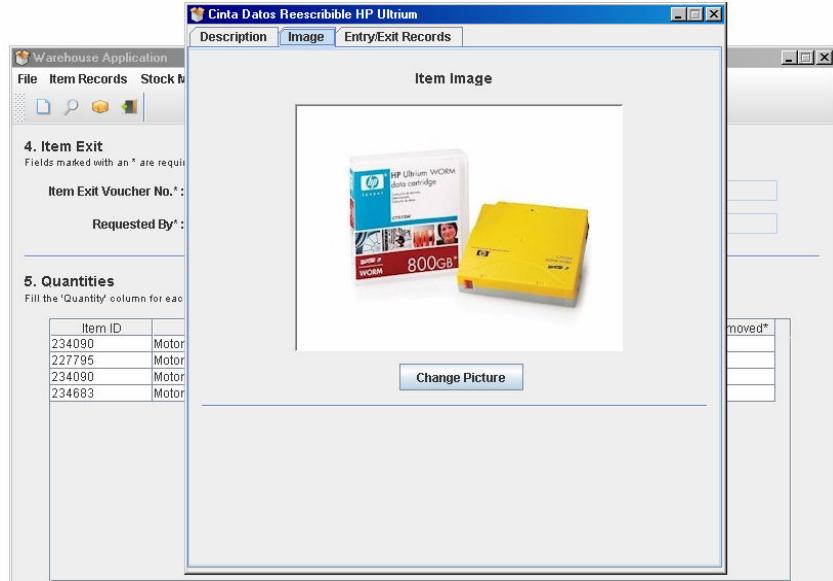
### Step 1: Quantities to Remove

This screen allows an employee to input the quantities of each item's unit of measurement to withdraw from the warehouse. These are the items that the user added to the list in the Multiple Item Search screen.

Item ID	Name	UM	In Stock	Quantity to be Removed*
234090	Motor Código: 301977	U	2	1
227795	Motor 230 VAC Código MT776X2A	U	6	3
234090	Motor Código: 301977	U	2	1
234683	Motor 0.75 kW para Molino Fino	U	0	0

#### User Comments:

The user approved the interface and arrangement. He requested that when an item is double-clicked, the item image should display.



## Step 2: Optimal Route

This screen displays the optimal pickup route for the items that are being withdrawn.

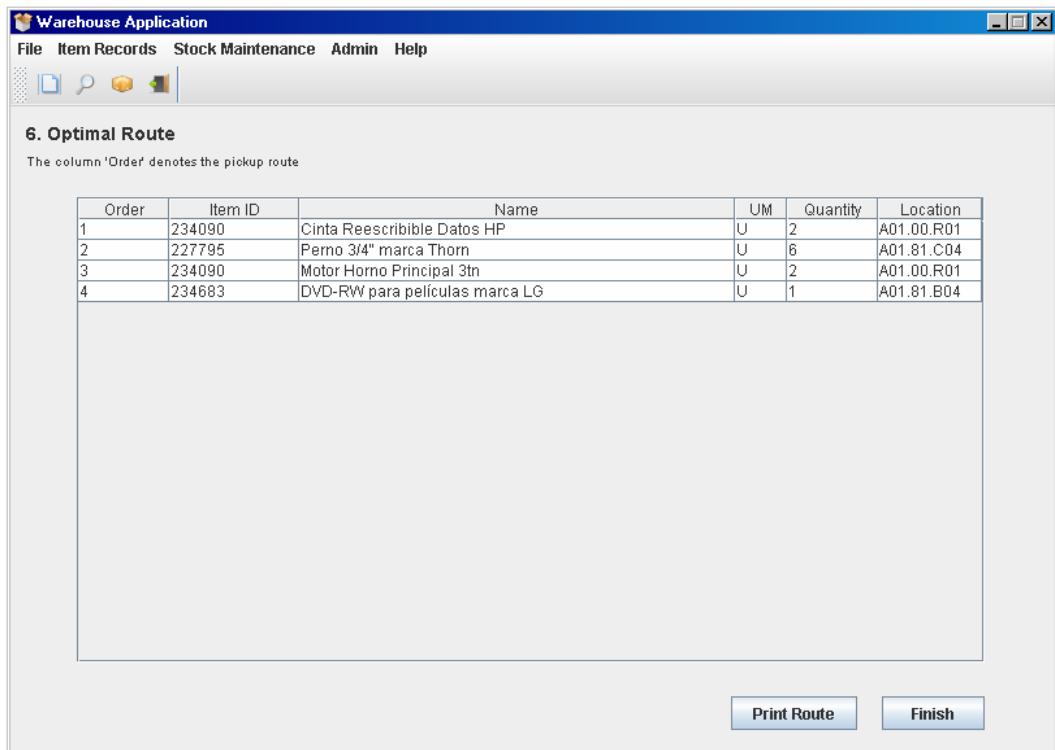
The screenshot shows the "Warehouse Application" window with the "Optimal Route" tab selected. The table lists the items to be removed with their respective locations:

Order	Item ID	UM	Quantity to be Removed	Location
1	234090	U	2	A01.00.R01
2	227795	U	6	A01.81.C04
3	234090	U	2	A01.00.R01
4	234683	U	1	A01.81.B04

Buttons at the bottom are "Print Route" and "Finish".

### User comments:

The user requested that the "Name" column be included in the table. A screenshot of the updated screen is shown below.



## Section B

---

# Detailed Design

## B1 – Data Structures

This section details the design of the main data structures to be used in the implementation of the solution.

Below are a series of class diagrams to illustrate the main attributes (instance variables) and operations (methods) for each data record class. A description of each is provided below. Only non-standard methods are outlined; getters and setters are implied for all classes.

### USERS

*Classes that deal with management of users logging into the application*

#### User

User	
-username : String	
-password : String	
-fullName : String	
-isAdmin : Boolean	
-isEntry : Boolean	
-isExit : Boolean	
-isEnabled : Boolean	
+setPassword() : void	

This class constructs the User object that stores a username, a password, the user's full name, permissions and current status. The class will mostly be used upon login, to fulfil objective 5.

Attribute	Description	Data Type	Sample Data
username	Unique identifier for a user	String	admin
password	Grants access to program	String	root1234
fullName	User's full name	String	System Administrator
admin	Gives user management permissions	boolean	false
entry	Allows a user to manage item entry operations	boolean	true
exit	Allows a user to manage item exit operations	boolean	false
userEnabled	Indicates if user is enabled to login to the system	boolean	true

Standard getter and setters will be included, as well as those that validate input to fulfil objective 5.2.3.

## ITEMS

*Classes that deal with item creation and data records that support this functionality*

### Group

Group
-code : short
-name : String
+toString() : String

This class constructs the Group object that holds information about a particular item group or family. It is an attribute of the Item object.

Attribute	Description	Data Type	Sample Data
code	The group's unique identifier	short	211
name	The group's name	String	Electronics

### UM (Unit of Measurement)

UM
-code : short
-name : String
+toString() : String

This class constructs the UM object that holds information about a unit of measurement with which an item is measured and stored inside the warehouse. It is an attribute of the Item object.

Attribute	Description	Data Type	Sample Data
code	The UM's unique identifier	short	2
name	The UM's name	String	Kilos (KLS)

### Location

Location
-warehouse : byte
-aisle : byte
-row : char
-column : byte
+toString() : String

This class constructs the Location object that holds information about where the item is stored inside the warehouse. It is an attribute of the Item object.

Attribute	Description	Data Type	Sample Data
warehouse	The code of the warehouse where the item is stored. This application handles items stored in warehouse 1 only, but this information is vital for accounting purposes.	byte	1

aisle	The aisle inside the warehouse where the item is stored	byte	23
row	The row of the aisle where the item is stored	char	C
column	The column of the aisle where the item is stored	byte	4

## Item

Item	
-ID : int -code : String -name : String -group : Group -um : UM -description : String -location : Location -createdBy : String -date : Date +writeItemToFile() : void	This class constructs the Item object that will hold all the information about items stored in the warehouse.

An item belongs to a group (of type Group), has a unit of measurement (of type UM) and is stored in a location (of type Location). These classes have been detailed above.

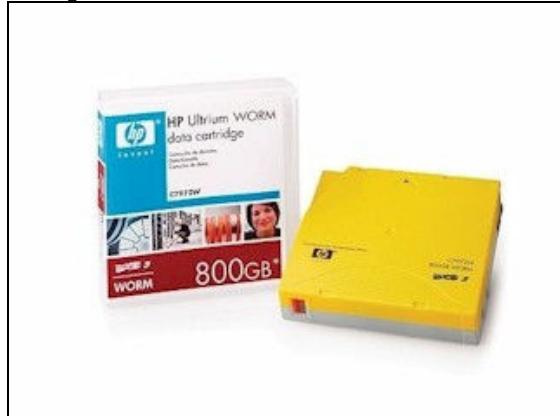
Attribute	Description	Data Type	Sample Data
ID	A unique integer identifier of the item in this system	int	1234
code	The item's code (matches the one from the existing system)	String	051243
name	The item's name	String	DVD Player
group	The group to which the item belongs. Sample data shows the output of the <code>toString()</code> method	Group	Systems
UM	The unit of measurement with which the item is stored. Sample data shows the output of the <code>toString()</code> method	UM	Kilos (KLS)
description	The item's description	String	Plays CDs, DVDs, DVD-RW. Color : Black.
location	The location inside the warehouse. Sample data shows the ouput of the <code>toString()</code> method	Location	A01.61.C12
createdBy	The username of the warehouse worker that created this item	String	moralea

date	The item's creation date	Date	03/07/2010
------	--------------------------	------	------------

*Other attributes:*

Image: the item's image, a 320 x 240 pixels .jpg image

Sample Data:



The image will be stored in the directory “..\\Files\\Item\_Images\\” and will have the item ID as its filename, with the .jpg extension. It will be loaded by checking if an image file with that filename exists, else the system assumes that this product has no image.

## Index

Index
-position : int
-field : String
+toString() : String

This class stores an index for an item. The field attribute stores the data by which the item is being indexed (it can be either by item name or code).

Attribute	Description	Data Type	Sample Data
position	The index's position in the Items File (equal to the Item ID)	int	31
field	The data by which the item is being indexed. Sample data shows an index by name.	String	DVD Player

The indexes will be used to have fast, direct access to the Items random access file. To allow for a fast and efficient search of items through name and code indexes, two binary trees will be populated when the application starts.

A binary tree is an Abstract Data Type (ADT) that has a tree structure where each node has a maximum of two children. This binary structure allows for binary search. As name and code are the most frequently used search criteria, the user will be offered fast and efficient search of the indexes for direct access to the items.

The indexes will be loaded onto an array in memory and sorted, and an algorithm will be implemented to add them to the binary trees in a way to maintain the tree balanced. A balanced tree has the advantage of not being skewed to one side, reducing the efficiency of the binary search.

A singularly linked list was also considered as an alternative, but searching would have to be linear, and the objective of implementing an ADT here is to achieve faster search. Because of this, a binary tree was the final choice.

Below are the classes necessary for the implementation of this ADT.

### IndexTNode

IndexTNode
-index : Index
-left : IndexTNode
-right : IndexTNode

A class that constructs the IndexTNode object, a node of a binary tree. It stores an Index object, that can either index items by name or by code.

Attribute	Description	Data Type	Sample Data
index	The indexed item	Index	31 "DVD Player" or 051243
left	The right child of this node	IndexTNode	Contains: - Index (sample data above)  Attributes of this class: - Left - Right

right	The left child of this node	IndexTNode	<p>Contains:</p> <ul style="list-style-type: none"> <li>- Index (sample data above)</li> </ul> <p>Attributes of this class:</p> <ul style="list-style-type: none"> <li>- Left</li> <li>- Right</li> </ul>
-------	-----------------------------	------------	---

## IndexBTree

IndexBTree
-root : IndexTNode
+isEmpty() : Boolean
+size() : int
+insertIndex() : void
+deleteIndex() : void
+getItem() : Item
+search() : Item
+getParent() : IndexTNode
+partialSearch() : ArrayList<Item>

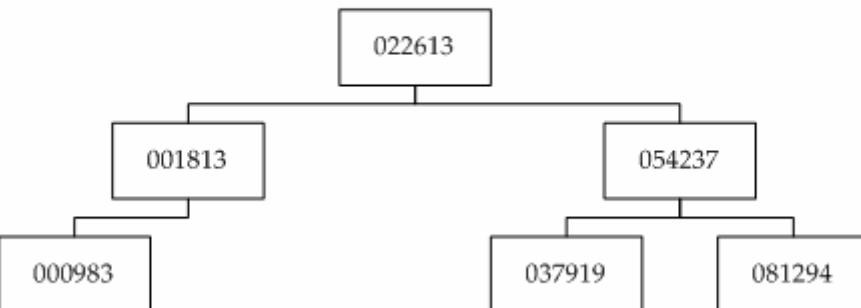
A class that constructs the IndexBTree object, a binary tree for of Indexes.

This class has functionality to support exact and partial searching of indexes.

Attribute	Description	Data Type	Sample Data
root	The root of the binary tree	IndexTNode	<p>Attributes:</p> <ul style="list-style-type: none"> <li>- Index (sample data above)</li> </ul> <p>Attributes of this class:</p> <ul style="list-style-type: none"> <li>- Left</li> <li>- Right</li> </ul>

Below are a few diagrams to illustrate some basic operations for binary trees. Each node represents an IndexTNode, which for this example contains an Index by code.

### Graphical Representation of a Binary Tree (by Item code)



### Adding an Item to a Binary Tree (by Item code)

The Item to add

012810

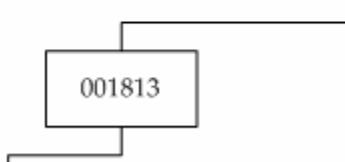
1. Look at Root - Is it larger or smaller?

If larger, move to right; if smaller, move to left



2. It is smaller. Move to left child. Is it larger or smaller?

If larger, move to right; if smaller, move to left

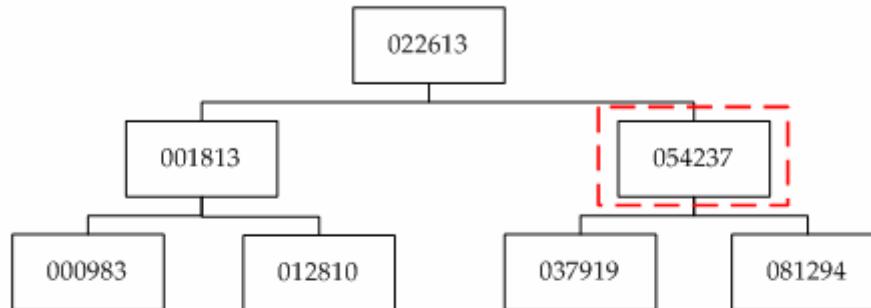


3. Continue to do this until a null child is reached. Add the item.



## Deleting an Item from a Binary Tree (by Item code)

The Item to delete - From now on "Item 1"

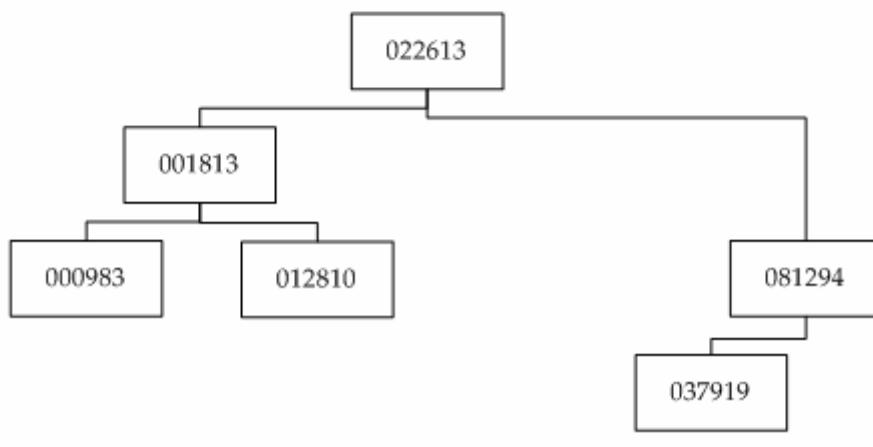


1. Make the Parent's right point to the Item 1's right



1. Make this item's left the Item 1's left

The Java Garbage Collector will delete Item 1



To search by item group or location, linear searching of the Items random access file will be implemented. This is not as efficient as searching a binary tree, but these criteria are the least frequently used.

## TRANSACTIONS

*Classes that deal with item processing (entry to or withdrawal from the warehouse)*

### TransactionRecord

TransactionRecord
- <code>ID</code> : int - <code>type</code> : int - <code>date</code> : Date - <code>quantity</code> : int - <code>balance</code> : int - <code>warehouseWorker</code> : String - <code>item</code> : Item - <code>document</code> : String - <code>thirdParty</code> : String + <code>toString()</code> : String + <code>processTransaction()</code> : void

A transaction record documents the entry to or withdrawal from the warehouse of an item. This class constructs the TransactionRecord object that stores this documentation.

This record applies for both an Entry transaction and an Exit transaction; the type of transaction is stored in the attribute type.

Attribute	Description	Data Type	Sample Data
<code>ID</code>	Transaction's unique identifier	int	21
<code>Type</code>	The type of transaction (Entry or Exit). A 0 denotes an item entry, a 1 denotes an item exit.	int	0
<code>date</code>	The date of the transaction.	Date	11/04/2010
<code>quantity</code>	Amount of the item's unit of measurement to be entered or removed	int	7
<code>balance</code>	The amount remaining of the item after this transaction	int	24
<code>warehouseWorker</code>	The username of the warehouse worker that processed this transaction.	String	moralea
<code>item</code>	The item that is being transacted. Sample data shows the item name.	Item	DVD Player

document	The document (Proof of Reception or Exit Voucher) that verifies this transaction.	String	00056212
thirdParty	Either the person/company that delivered the items or the person that requested the item and withdraws it	String	Omnia Solutions S.A.C.

The most common operation in the program will be to record a transaction, either an item entry or exit. Even though the number of items may be very large, the number of transactions will be even larger, as there might be hundreds of transactions per item going back several years.

Transactions may be stored in a static ADT such as a hash table, but given the frequent addition of new transactions, to keep from having to constantly grow the array's length, a dynamic ADT would be more appropriate.

I have chosen a doubly linked list to fulfil these objectives. As transactions have to be ordered descending by date, as requested by the user, every new transaction will be added to the head of the list. The list is doubly linked to allow for efficient deletion of transactions. The following classes are necessary to implement the linked list:

### TransactionNode

<b>TransactionNode</b>	A class that constructs the TransactionNode object, a node of the linked list. It stores the transaction and a pointer to the next and previous transactions.
<pre>-transaction : TransactionRecord -next : TransactionNode -prev : TransactionNode</pre>	

<b>Attribute</b>	<b>Description</b>	<b>Data Type</b>	<b>Sample Data</b>
transaction	The transaction stored in the list's node	TransactionRecord	<i>Sample data shown above</i>
next	The next transaction. Ordered descending by date	TransactionNode	Contains : - Transaction - Prev - Next
prev	The previous transaction. Ordered descending by date	TransactionNode	Contains : - Transaction - Prev - Next

## TransactionList

TransactionList
-head : TransactionNode
+addToHead() : void
+size() : int
+isEmpty() : Boolean
+searchTransactions() : ArrayList
+deleteTransaction() : void

This class constructs the object that holds the list of transactions. As every node has a pointer to the next transaction, the list is created by creating a pointer to the head of the list.

Attribute	Description	Data Type	Sample Data
head	The head of the linked list (first item)	TransactionNode	<i>Sample data above</i>

The following are some diagrams to illustrate the implementation of the Transactions list. The data stored in each node only shows the transaction ID.

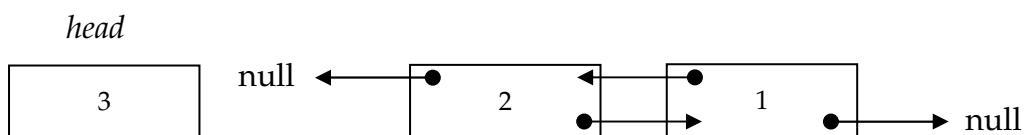
### Graphical representation of the Transactions list



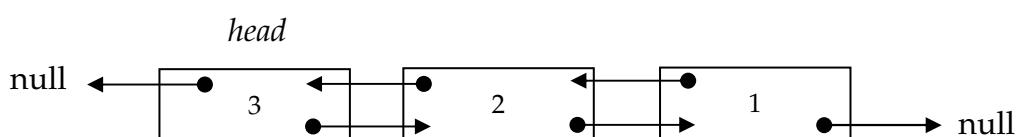
As transactions are stored by date in descending order, there is no need to add a node to the tail of the list, as the first transaction will always be the latest one. This is also why the addBefore() and addAfter() methods would be redundant.

### Adding a node to the head of the list

#### 1. Create a new TransactionNode and set it as the head



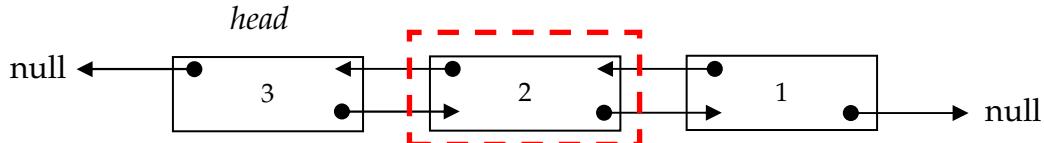
#### 2. Make the new head's next point to the old head Make the old head's prev point to the new head



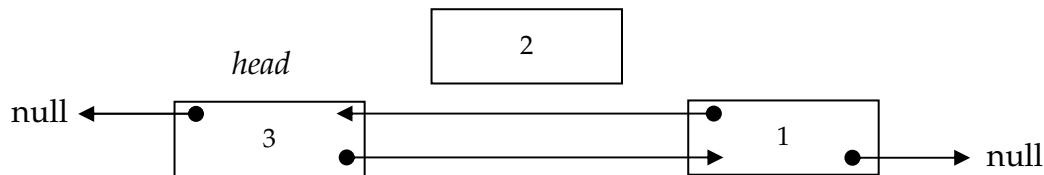
As these transaction records have to be up-to-date and precise, there is no reason why a regular user would want to delete a transaction. This could compromise the integrity of the tax report presented to a government organization. A `deleteNode()` method will be implemented and accessible only to admin users.

### Deleting a node from the list

#### 1. The node to delete



#### 2. Make the next of the node's prev point to the node's next Make the prev of the node's next point to the node's prev



#### 3. Set the node's prev and next to be null The Java Garbage Cleaner will dispose of it

## FILES

Data has to be stored in secondary storage so that the scope of the data and operations is not limited to runtime. The three main data to be stored in secondary storage are users, items and transactions.

### Users

The program will not have a large number of users using it; the number of users will most probably not surpass the 50 mark. Because of this, the need for ultra-fast access to this data by Random Access Files is not required. A sequential file will be used to store user information. The file will be pipe (|) delimited. A pipe will be used as it is not a character that is commonly used in naming. Booleans will be stored as integers: a 0 denotes false, a 1 denotes true.

Every record will have the following format:

username | password | fullName | isAdmin | isEntry | isExit | isEnabled

The file will be ordered alphabetically by username. This will guarantee an efficient search. If a username starting with 'c' is being searched, and the record being read starts with the letter 'd', then the user has not been found, and the other records do not need to be read.

*Sample Data:*

```
admin | root | System Administrator | 1 | 1 | 1 | 1  
locks | pass456 | Santiago Lock | 0 | 0 | 1 | 0  
moralea | 1234 | Alvaro Morales | 0 | 1 | 0 | 1
```

**Items**

A list of items will be stored in a Random Access File, in ascending order by ID. A Random Access File is appropriate as because the file will get very large, sequential rather than direct access would make the accessing of items very inefficient.

Field	Data Type	No. Bytes	Sample Data
ID	int	4	000491
code	String	12	042912
name	String	400	DVD Player
code (group)	short	2	211
code (UM)	short	2	086
description	String	400	Plays DVD-R, DVD-RW
warehouse (location)	byte	1	01
aisle (location)	byte	1	23
column (location)	byte	1	4
row (location)	char	2	C
createdBy	String	40	moralea
year	short	2	2009
month	byte	1	6
dayOfMonth	byte	1	29

Though it seems that the name and description fields take up a lot of memory, this was done because the user specifically requested the 200 character length limit.

This gives a total record length of **869 bytes**.

The item image will be stored in a directory and will have as its name the item's ID

E.g.: ...\\Images\\491.jpg

The file will be ordered by ID. The user needs to search for items based on code, name, group and location.

Index files for code and name will be stored, so as to search the index files and have random access to the Items file. An index stores the field by which the item is being indexed (name or code) and its position in the Items random access file. Using the seek() method, the indexed record may be accessed directly.

#### *The Item\_Name Index*

The file will be an ordered sequential file sequential file, delimited by pipes (|).

It will store the name of the item and its location in the random access file.

name | locationInFile

#### *Sample Data:*

DVD Player | 83  
Grain Windmill Bolts | 21  
Heat-resistant Bricks | 45

#### *The Item\_Code Index*

The file will be an ordered sequential file sequential file, delimited by pipes (|).

It will store the code of the item and its location in the random access file.

code | locationInFile

#### *Sample Data:*

000231 | 83  
068495 | 21  
078246 | 45

### **Transactions**

As an item can have many transactions, and transaction information is only relevant to one item, the records of transactions for each item will be stored in separate files.

The files will be named after the item's ID with the following folder and naming structure:

...\\Transactions\\ItemID.txt

*Sample data*

...\\Transactions\\26102.txt  
...\\Transactions\\67119.txt

This will be stored in an ordered sequential file that will be pipe-delimited (|). Each record will have the following format:

ID | type | day | month | year | quantity | balance | warehouseStaff | document | third-party

The type field stores the type of transaction; a 0 denotes an item entry, a 1 denotes an item exit. This means that the document will either be the Proof of Reception or Exit Voucher, and the third-party will either be the person that requested the item or the person/company that delivered the item.

*Sample Data:*

1 | 23 | 10 | 2008 | 3 | 10 | Alvaro Morales | 000321 | Santiago Lock

New transactions will be added to the end of the file. As the user needs to see the transactions in descending order by date, each record will be added to the head of the transactions list, so that the latest transaction will be the head.

---

## B2 - ALGORITHMS

---

This section describes the algorithms that will be coded to fullfil the design requirements outlined below. Algorithms are grouped by general functionality (specific function and operation information is described in the comments).

### ALGORITHM LIST

<b>USER MANAGEMENT .....</b>	<b>68</b>
<i>Perform login .....</i>	68
<i>Create a user from a string in the Users file.....</i>	68
<i>Convert an int to a boolean.....</i>	69
<i>Change user's password in the Users file .....</i>	69
<i>Update a user in the Users file.....</i>	70
<i>Create a new user in the Users file .....</i>	71
<i>Delete a user from the Users file .....</i>	71
<b>ITEM HANDLING .....</b>	<b>72</b>
<i>Write a new item to the Items file.....</i>	72
<i>Write a new Item index to the appropriate index file .....</i>	73
<i>Read indexes into an array .....</i>	74
<i>Create a balanced binary tree of indexes .....</i>	75
<i>Delete an item .....</i>	76
<i>Delete an index by code from the file .....</i>	77
<i>Delete an index by name from the file.....</i>	78
<i>Update an item.....</i>	78
<i>Update an item's index by code .....</i>	79
<i>Update an item's index by name.....</i>	80
<i>Get an item from the Items file .....</i>	81
<i>Search for an item by group.....</i>	83
<i>Search for an item by location.....</i>	84
<i>Calculate the optimal route of an item pickup order.....</i>	85
<b>OPERATIONS OF THE BINARY TREE OF ITEM INDEXES .....</b>	<b>86</b>
<i>Insert a node into the tree .....</i>	86

<i>Get a node from the binary tree given its field</i> .....	87
<i>Get a node's parent</i> .....	88
<i>Search for a node by an exact query</i> .....	89
<i>Search for a node by a partial query</i> .....	90
<i>Delete a node from the tree</i> .....	93
<b>TRANSACTIONS HANDLING</b> .....	<b>94</b>
<i>Get a transaction from the Transactions file</i> .....	94
<i>Get the quantity in stock of an item</i> .....	95
<i>Process an entry or exit transaction</i> .....	95
<b>OPERATIONS OF THE LINKED LIST OF TRANSACTIONRECORDS</b> .....	<b>96</b>
<i>Add a TransactionNode to the head of the list</i> .....	96
<i>Search for TransactionRecords between specified dates</i> .....	97
<i>Delete a TransactionNode</i> .....	98

# USER MANAGEMENT

## Perform login

### Params:

- username: the text input from the username JTextField
- password: the text input from the password JPasswordField

### Returns:

- The user that logged in

```
public User login(String username, String password){  
  
    User user;  
  
    Read Users File  //(stored in ...\\Users.txt)  
    While the end of the file has not been reached {  
        Store the line being read in a String  
        Create variable 'user' with that String  
            //using method createUser(String), outlined in pages 67-68  
  
        If username and password match user.getUsername() and user.getPassword  
            return user  
        Else  
            Read the next line  
  
    return null      //if the user is not found
```

## Create a user from a string in the Users file

### Params:

```
- line: a tokenized String
Returns:
- a User object

public User createUser(String line) {

    Separate tokens from line with the token "|"
        //The user is stored in the file with the format:
        //username|password|fullName|isAdmin|isEntry|isExit|isEnabled
    Create and return a new User with the tokens as params.
        //For permissions (where the tokens are ints), use the method intToBoolean(), outlined below
}
```

## Convert an int to a boolean

```
Params:
- i: an int (either 1 or 0)
Returns:
- true if i is 1
- false if i is 0
```

```
public boolean intToBoolean(int i){
    If i is 1, return true, otherwise return false
}
```

## Change user's password in the Users file

```
Params:
- newPassword: the new password of the user that logged in
Returns: void

public void performPasswordChange(String newPassword){
```

```
this.user.setPassword(newPassword);
Read Users File  //(stored in ...\\Users.txt)
Create temporary file (Users_tmp.txt)
For every record in the Users file
    If the username of the record being read matches the username of the user that logged in
        Write this.user.toString() to temporary file  //Will tokenize user with '|'
    Else
        Write record in the Users file to the temporary file

Delete Users.txt
Rename Users_tmp.txt to Users.txt

}
```

## Update a user in the Users file

**Params:**

- user: the user being updated

**Returns:** void

```
public void updateUser(User user){

    Read Users File  //(stored in ...\\Users.txt)
    Create temporary file (Users_tmp.txt)
    For every record in the Users file
        If the username of the record being read matches the username of the user that has been updated
            Write user.toString() to temporary file  //Will tokenize user with '|'
        Else
            Write record in the Users file to the temporary file

    Rename Users_tmp.txt to Users.txt
}
```

## Create a new user in the Users file

**Params:**

- user: the user that was created in memory

**Returns:** void

```
public void createUser(User user){  
  
    Read Users File    //(stored in ...\\Users.txt)  
    Create temporary file (Users_tmp.txt)  
  
    For every record in the Users file  
        Create a user from the record read //by calling createUser() with the line read  
        If the username of the user being created is less than the username of the record being read  
        //Using compareTo()  
            Write user.toString() to temporary file  
            Write record read to temporary file  
        Else  
            Write record read to temporary file  
  
    Delete Users.txt  
    Rename Users_tmp.txt to Users.txt  
  
}
```

## Delete a user from the Users file

**Params:**

- user: the user that will be deleted

**Returns:** void

```
public void deleteUser(User user){  
  
    Read Users File      //(stored in ...\\Users.txt)  
    Create temporary file //\\Users_tmp.txt)  
  
    For every record in the Users file  
        If the username of the user being deleted does NOT match the username of the record being read  
            Write record to temporary file  
  
    Delete Users.txt  
    Rename Users_tmp.txt to Users.txt  
  
}
```

## ITEM HANDLING

### Write a new item to the Items file

#### Params:

- item: the item that is being created

#### Returns: void

```
public void createItem(Item item){  
  
    Create a RandomAccessFile      //(from the file ..\\Items.dat)  
  
    Seek to the first available empty position  
    Seek to the start of the item before it  
    Read Item ID of the item and store in int 'ID'  
    Seek to the end of that item  
  
    //This is all for the item being created  
    Write the Item ID: writeInt(ID + 1) //the internal item ID
```

```
Write the code: writeInt(item.getCode())
String name = item.getName()
While the length of name is not 200 characters
    Append spaces to end of name
Write the name: writeUTF(name)
Write the group code: writeInt(item.getGroup().getCode())
Write the UM's code: writeInt(item.getUM().getCode())
String description = item.getDescription()
While the length of description is not 200 characters
    Append spaces to end of description
Write the description: writeUTF(description)
Write the warehouse number: writeByte(1) //This application only deals with items in warehouse 1
Write the aisle number: writeByte(item.getLocation().getAisle())
Write the column number: writeChar(item.getLocation().getColumn())
Write the row number: writeByte(item.getLocation().getRow())

//Create records to add to index files (Internal ID gives position in Random Access File)
CodeIndex code = new CodeIndex(item.getCode, item.getID)
NameIndex name = new NameIndex(item.getName, item.getID)

Call the writeIndex() method with 'code' and indexType by code
Call the writeIndex() method with 'name' as param //outlined in pages 72-73

Add 'code' to the Item Code tree
Add 'name' to the Item Name tree //using insert(), outlined in pages 85-86

}
```

## Write a new Item index to the appropriate index file

### Params:

- index: the index that will be created
- indexType: the type of index (by code or by name)

### Returns: void

```
public void writeIndex(Index index, int indexType){

    if index == index by code          //indexTypes stores as int constants
        Read the index file of Item codes //stored in ...\\Item_Code.txt
        Create a temporary file (Item_Code_Tmp.txt)

        For every record in the file
            Write the record to the temporary file

        Write index.toString() to the end of the file //makes the user a tokenised string

        Delete Item_Code.txt
        Rename Item_Code_Tmp.txt to Item_Code.txt

    else      //index is an index by name
        Read the index file of Item names //stored in ...\\Item_Name.txt
        Create a temporary file (Item_Name_Tmp.txt)

        For every record in the file
            Write the record to the temporary file

        Write index.toString() to the end of the file //makes the user a tokenised string

        Delete Item_Name.txt
        Rename Item_Name_Tmp.txt to Item_Name.txt

}
```

## Read indexes into an array

**Params:**  
- file: the index file  
**Returns:** void

```
public Index[] readIndexes(File file){  
  
    int numberOfRecords = 0;  
  
    While the end of the file has not been reached  
        numberOfRecords++;  
  
    Index[] indexes = new Index[numberOfRecords]  
  
    For every record in the file  
        Create an Index from the record being read and add it to the array  
  
    Sort the array (using Arrays.sort() method)  
  
    Return indexes  
  
}
```

## Create a balanced binary tree of indexes

Method makes a recursive call to `createIndexBinaryTree()` to make sure that the items of the ordered array are added in a way so that the binary tree of indexes is properly balanced

**Params:**

- `indexes`: an ordered `Index[]`
- `tree`: the binary tree of indexes

**Returns:** void

```
public void createIndexBinaryTree(Index[] indexes, IndexBTree tree){  
  
    createIndexBinaryTree(names, 0, names.length)  
  
}
```

Method is called recursively to make sure that the items of the ordered array are added in a way so that the binary tree of indexes is properly balanced

**Params:**

- indexes: an ordered Index[]
- tree: the binary tree of indexes
- start: a pointer of the start of the array block
- finish: a pointer of the end of the array block

**Returns:** void

```
public void createIndexBinaryTree(Index[] indexes, IndexBTree tree, int start, int finish){  
  
    int mid = ((finish-start)/2) + start  
    int size = finish - start  
  
    if mid > indexes.length-1  
        Do nothing  
    else if size equals 0  
        Insert indexes[mid] to binary tree using insert()          //outlined in page 86  
    else  
        Insert indexes[mid] to binary tree using insert()  
        createIndexBinaryTree(indexes, tree, start, mid)  
        createIndexBinaryTree(indexes, tree, mid+1, finish)  
  
}
```

## Delete an item

**Params:**

- item: the item to delete

**Returns:** void

```
public void deleteItem(Item item){  
  
    Create a RandomAccessFile      //(from the file ..\Items.dat)
```

```
Seek to (item.getID * RECORD_LENGTH)      //will move to where the record to delete is stored
Delete item by writing ID to value '-999' //denotes that record is empty

deleteNameIndex(new NameIndex(item.getName(), item.getID()))           //Delete the item from the name index file
deleteCodeIndex(new CodeIndex(item.getCode(), item.getID()))           //Delete the item from the code index file

Delete from NameIndex tree
Delete from CodeIndex tree    //by using deleteIndex() outlined in page 93

}
```

## Delete an index by code from the file

### Params:

- item: the item to delete

### Returns: void

```
public void deleteCodeIndex(Item item) {

    Read Code Index File          //stored in ...\\Item_Code.txt
    Create temporary file         //Item_Code_Tmp.txt

    For every record in the Code Index file
        If the code and ID of the item being deleted does NOT match the code and ID of the index being read
            Write record to temporary file

    Delete Item_Code.txt
    Rename Item_Code_Tmp.txt to Item_Code.txt

}
```

## Delete an index by name from the file

**Params:**

- item: the item to delete

**Returns:** void

```
public void deleteNameIndex(Item item){  
  
    Read Name Index File      //stored in ...\\Item_Name.txt  
    Create temporary file    //Item_Name_Tmp.txt  
  
    For every record in the Name Index file  
        If the name and ID of the item being deleted does NOT match the name and ID of the index being read  
            Write record to temporary file  
  
    Delete Item_Name.txt  
    Rename Item_Name_Tmp.txt to Item_Name.txt  
  
}
```

## Update an item

**Params:**

- oldCode: the previous code
- oldName: the previous name
- item: the updated item

**Returns:** void

```
public void updateItem(String oldCode, String oldName, Item item){  
  
    Create a RandomAccessFile    //from the file ..\\Items.dat  
    Seek to (item.getID * RECORD_LENGTH)
```

```
//This is all for the item being updated
Write the Item ID: writeInt(item.getID())      //the internal item ID
Write the code: writeInt(item.getCode())
String name = item.getName()
While the length of name is not 200 characters
    Append spaces to end of name
Write the name: writeUTF(name)
Write the group code: writeInt(item.getGroup().getCode())
Write the UM's code: writeInt(item.getUM().getCode())
String description = item.getDescription()
While the length of description is not 200 characters
    Append spaces to end of description
Write the description: writeUTF(description)
Write the warehouse number: writeByte(1)  //This application only deals with items in warehouse 1
Write the aisle number: writeByte(item.getLocation().getAisle())
Write the column number: writeChar(item.getLocation().getColumn())
Write the row number: writeByte(item.getLocation().getRow())

updateCodeIndexFile(oldCode, item)          //update the index by code, outlined in pages 79-80
updateNameIndexFile(oldName, item)          //update the index by name, outlined in pages 80-81

Delete the index by name of item from the Name Index Binary Tree
Delete the index by code of item from the Code Index Binary Tree      //using delete outlined in page 93

Index code = new Index(item.getCode(), item.getID())
Index name = new Index(item.getName(), item.getID())

Add 'code' to the Item Code tree
Add 'name' to the Item Name tree      //using insert() outlined in page 86

}
```

## Update an item's index by code

**Params:**

- oldCode: the previous code
- item: the updated item

**Returns:** void

```
public void updateCodeIndexFile(String oldCode, Item item){  
  
    Read Code Index File          //stored in ...\\Item_Code.txt  
    Create temporary file        //Item_Code_Tmp.txt  
  
    For every record in the Code Index file  
        If the code of the index being read does NOT match oldCode  
            Write record to temporary file  
  
        Else  
            Write (new Index(item.getCode(), item.getID())).toString() to file  
  
    Delete Item_Code.txt  
    Rename Item_Code_Tmp.txt to Item_Code.txt  
  
}
```

## Update an item's index by name

**Params:**

- oldName: the previous name
- item: the updated item

**Returns:** void

```
public void updateCodeIndexFile(String oldName, Item item){  
  
    Read Name Index File          //stored in ...\\Item_Name.txt  
    Create temporary file        //Item_Name_Tmp.txt
```

```
For every record in the Name Index file
    If the code of the index being read does NOT match oldCode
        Write record to temporary file

    Else
        Write (new Index(item.getName(), item.getID())).toString() to file

Delete Item_Name.txt
Rename Item_Name_Tmp.txt to Item_Name.txt

}
```

## Get an item from the Items file

**Params:**

- position: the position of the item in the file

**Return:**

- the item at that position

```
public Item getItemFromFile(int position){

    Create a RandomAccessFile      //from the file ..\Items.dat
    file.seek(position * RECORD_LENGTH)

    int ID = file.readInt()

    If (ID == position)
        String code = file.readUTF()
        String name = file.readUTF()
        short groupCode = file.readShort()
        short umCode = file.readShort()
        String description = file.readUTF()
        byte warehouseNumber = file.readByte()
        byte aisleNumber = file.readByte();
```

```
byte columnNumber = file.readByte();
char row = file.readChar();
String createdBy = file.readUTF();
short year = file.readShort();
byte month = file.readByte();
byte day = file.readByte();

Group group = getGroup(groupCode)
UM um = getUM(umCode)

Location location = new Location(warehouseNumber, aisleNumber, columnNumber, row);
GregorianCalendar creationDate = new GregorianCalendar(year, month, day);

return new Item(ID, code, name, group, um, description, location, createdBy, creationDate);
}
```

Gets a group from the groups array

**Params:**

- groupCode: a group's code

**Return:**

- an item group with that code

```
public static Group getGroup(short groupCode){
    for every item in the Groups array{           //a static array of groups loaded at runtime
        if(the code of the group at that position matches groupCode){
            return that group;
        }
    }

    return null;
}
```

Gets a UM from the UM array

**Params:**

- umCode: a group's code

**Return:**

- a UM with that code

```
public static Group getUM(short umCode){  
    for every item in the UM array{           //a static array of UMs loaded at runtime  
        if(the code of the UM at that position matches umCode){  
            return that UM;  
        }  
    }  
  
    return null;  
}
```

## Search for an item by group

### Params:

- groupCode: the group's code, the search criteria

### Returns:

- An array list of items matching the search criteria

```
public ArrayList searchByGroup(short groupCode){  
  
    ArrayList list = new ArrayList()  
    Create a RandomAccessFile      //from the file ..\Items.dat  
  
    int position;  
    int numberOfRecords = file.length/RECORD_LENGTH  
  
    For (int i=0;i<numberOfRecords;i++)  
        File.seek(i * RECORD_LENGTH)  
        position = file.readInt()  
        if(position != -999)          //if record is not empty  
            Item item = getItemFromFile(position)  
            If (item.getGroup().getCode() == groupCode)  
                Add it to the ArrayList
```

```
list.trimToSize()
return list
}
```

## Search for an item by location

**Params:**

- location: a location in the warehouse

**Returns:**

- An array list of items matching the search criteria

```
public ArrayList searchByLocation(Location location){

    ArrayList list = new ArrayList()
    Create a RandomAccessFile      //from the file ..\Items.dat

    int position;
    int numberOfRecords = file.length/RECORD_LENGTH

    For (int i=0;i<numberOfRecords;i++)
        File.seek(i * RECORD_LENGTH);
        position = file.readInt()
        if(position != -999)          //if record is not empty
            Item item = getItemFromFile(position)
            If (item.getLocation.getAisle() == location.getAisle &&
                item.getLocation.getRow == location.getRow &&
                item.getLocation.getColumn == location.getColumn)
                Add it to the ArrayList

    list.trimToSize()
    return list
}
```

## Calculate the optimal route of an item pickup order

**Params:**

- itemsToProcess: an ArrayList of the items to withdraw

**Returns:**

- an ArrayList of items in the pickup order

```
public ArrayList getOptimalRoute(ArrayList itemsToProcess){  
  
    if (itemsToProcess.size() == 1)  
        return itemsToProcess  
  
    else  
        ArrayList group1 = new ArrayList()  
        ArrayList group2 = new ArrayList()  
  
        For every item in itemsToProcess  
            Item currentItem = The item of itemsToProcess in that position  
  
            If(currentItem.getLocation().getAisle() <= 36)  
                group1.add(currentItem)  
            Else  
                group2.add(currentItem)  
  
        group1.trimToSize()  
        Sort group1 by location using the Collections.sort() method  
  
        group2.trimToSize()  
        Sort group2 by location using the Collections.sort() method  
  
        ArrayList route = new ArrayList()  
  
        For every item in group1
```

```
    route.add(group1.get(item at that position)

For every item in group2
    route.add(group2.get(item at that position)

route.trimToSize()
return route

}
```

## OPERATIONS OF THE BINARY TREE OF ITEM INDEXES

### Insert a node into the tree

Inserts an Index to the tree by recursively calling the method insertIndex(IndexTNode current, IndexTNode index)

**Params:**

- index: the Index to insert

**Returns:** void

```
public void insertIndex(IndexTNode index) {

    if the tree is empty
        //add dummy root to avoid the root (and consequently the entire tree) being deleted
        create a dummy root with the first letter of the data in index
        assign the root of the tree to be the dummy root
        insertIndex(index)
    else
        insertIndex(root, index)

}
```

Method is called recursively to insert an index to the Index binary tree

**Params:**

```
- current: a pointer to the current node being evaluated for insertion
- index: the node to insert
Returns: void

public void insertIndex(IndexTNode current, IndexTNode index) {
    if the data in index matches the data in current      //already exists
        do nothing
    else if the data in index is less than the data in current //using compareTo()
        if current has no left child
            assign index as current's left child
        else
            insertIndex(current.getLeft(), index)
    else if the data in index is greater than the data in current      //using compareTo()
        if current has no right child
            assign index as current's right child
        else
            insertIndex(current.getRight(), index)
}
```

## Get a node from the binary tree given its field

Gets a node from the binary tree given its field, by recursively calling the method getNode(IndexTNode current, String field)

**Params:**

- field: a node's field (item code or name)

**Returns:**

- a node containing that field, or null if not found

```
public IndexTNode getNode(String field) {
    return getNode(root, field)
}
```

Method is called recursively to get a node from the binary tree given its field

**Params:**

```
- current: a pointer to the current node
- field: a node's field (item code or name)
Returns:
- a node containing that field, or null if not found

public IndexTNode getNode(IndexTNode current, String field) {
    if current == null
        return null
    else
        if the data in current is greater than field           //using compareTo()
            return getNode(current.getLeft(), field)
        else if the data in current is less than field       //using compareTo()
            return getNode(current.getRight(), field)
        else
            return current
}
```

## Get a node's parent

Gets the parent of a node by recursively calling the method getParent(IndexTNode current, IndexTNode node, String field)

**Params:**
- node: the child node

**Returns:**
- the node's parent

```
public IndexTNode getParent(IndexTNode node){
    return getParent(root, node, node.getIndex().getField());
}
```

Method is called recursively to find a node's parent

**Params:**
- current: a pointer to the current node being evaluated
- node: the child node

```
- field: the data that the node stores
Returns:
- the node's parent

public IndexTNode getParent(IndexTNode current, IndexTNode node, String field) {

    if (current == null)
        return null
    else if the data stored in current matches field and it is the root of the tree
        return null      //root has no parent
    else if the node has a left child and the data stored in the left child matches field
        return current
    else if the node has a right child and the data stored in the right child matches field
        return current
    else if the data in current is less than field          //using compareTo()
        return getParent(current.getRight(), node, field)
    else if the data in current is greater than field      //using compareTo()
        return getParent(current.getLeft(), node, field)

}
```

## Search for a node by an exact query

Searches the binary tree for a node exactly matching a query, by making a recursive call to the method  
`search(IndexTNode current, String query)`

**Params:**

- query: the exact match for an item name or code

**Returns:**

- a node matching that query, or null if no results are found

```
public IndexTNode search(String query) {
    return search(root, query)
}
```

Traverses the binary tree to find a node that exactly matches a query

**Params:**

- current: a pointer to the current node
- query: the exact match for an item name or code

**Returns:**

- a node matching that query, or null if no results are found

```
public IndexTNode search(String query) {  
    if current == null  
        return null  
    else  
        if the data stored in current is greater than query //using compareToIgnoreCase()  
            return search(current.getLeft(), query)  
        else if the data stored in current is less than query //using compareToIgnoreCase()  
            return search(current.getRight(), query)  
        else if the data stored in current equals query //using compareToIgnoreCase()  
            return getItemFromFile(current.getIndex().getPosition())  
    }  
}
```

## Search for a node by a partial query

Searches the binary tree for nodes partially matching a query, by getting and trimming a sub-tree

**Params:**

- query: the starting characters of an item name or code

**Returns:**

- an ArrayList of search results; null if no results found

```
public IndexTNode partialSearch(String query) {  
    IndexBTree subTree = getPartialTree(query)  
  
    if subTree == null  
        return null  
    else
```

```
        return getResults(partialTree, query)
    }
```

Gets a sub-tree from the first node that matches the search criteria, by calling the method  
`getMatchingSubTree(IndexTNode current, String field)`

**Params:**

- `query`: the starting characters of an item name or code

**Returns:**

- an `IndexBTree` that has as its root the first node that matches the query; null if no results found

```
public IndexBTree getPartialTree(String query) {
    return getMatchingSubTree(root, query)
}
```

Method is called recursively to get a sub-tree from the first node that matches the search criteria

**Params:**

- `current`: a pointer to the current node
- `query`: the starting characters of an item name or code

**Returns:**

- an `IndexBTree` that has as its root the first node that matches the query; null if no results found

```
public IndexBTree getMatchingSubTree(IndexTNode current, String query) {
    if current == null
        return null

    if the data stored in current equals query      //using compareToIgnoreCase()
        return new IndexBTree(current)           //with current as the root
    else if the data stored in current starts with query
        return new IndexBTree(current)
    else if the data stored in current is less than query
        return getMatchingSubTree(current.getRight(), query)
    else if the data stored in current is greater than query
        return getMatchingSubTree(current.getLeft(), query)
    else
        return null
}
```

Returns an ArrayList of items that match the query

**Params:**

- tree: a sub-tree matching that has the first node that matched the query as its root
- query: the starting characters of an item name or code

**Returns:**

- an ArrayList of items that match the query; null if no results found

```
private ArrayList getResults(IndexBTree tree, String query) {  
    ArrayList searchResults = new ArrayList()  
    addResults(tree.getRoot(), query, searchResults)  
  
    if search results is not empty  
        searchResults.trimToSize()  
        return searchResults  
    else  
        return null  
}
```

Traverses the tree and adds nodes that match the query to the ArrayList of search results

**Params:**

- current: a pointer to the current node
- query: the starting characters of an item name or code
- searchResults: an ArrayList of items matching the query

**Returns:** void

```
public void addResults(IndexTNode current, String query, ArrayList searchResults){  
    if current is null  
        do nothing  
    else if current is the dummy root  
        addResults(current.getLeft(), query, searchResults)  
        addResults(current.getRight(), query, searchResults)  
    else if the data in current does NOT start with query  
        if current has a left child and the data in the left child starts with query  
            addResults(current.getLeft(), query, searchResults)  
        else if current has a right child and the data in the right child starts with query
```

```
        addResults(current.getRight(), query, searchResults)
    else
        do nothing
else
    addResults(current.getLeft(), query, searchResults)
Add the item indexed by current to the ArrayList      //using getItemFromFile()
    addResults(current.getRight(), query, searchResults)
}
```

## Delete a node from the tree

**Params:**

- index: the node to delete

**Returns:** void

```
public void deleteIndex(IndexTNode index){
    if the node has no children
        setRoot(null)
    else
        String s = getParent(node).getIndex().getField()
        If node is the right child of its parent
            if(node.getRight() != null)
                getParent(node).setRight(node.getRight())

            if(node.getLeft() != null)
                node.getRight().setLeft(node.getLeft());

            if(node.getLeft() == null && node.getRight() == null)
                getParent(node).setRight(null)

            node.setLeft(null)
            node.setRight(null)
```

```
else //If it is the left child of its parent
    if (node.getLeft() != null)
        getParent(node).setLeft(node.getLeft())

    if (node.getRight() != null)
        node.getLeft().setRight(node.getRight())

    if (node.getLeft() == null && node.getRight() == null){
        getParent(node).setLeft(null)

        node.setLeft(null)
        node.setRight(null)

    }
```

## TRANSACTIONS HANDLING

### Get a transaction from the Transactions file

**Params:**

- line: a tokenized String read from the Transactions file

**Returns:**

- a TransactionRecord object

```
public TransactionRecord readTransaction(Item item, String line) {

    Separate tokens from line with the token "|"
        //The transaction record is stored in the file with the format:
        // ID|type|day|month|year|quantity|balance|warehouseStaff|document|third-party
    Create and return a new TransactionRecord with the tokens as params.
}
```

## Get the quantity in stock of an item

**Params:**  
- item: the item to find its quantity in stock

**Returns:**  
- an int with the quantity of the item's UM in stock

```
public int getQuantityInStock(Item item) {
    int ID = item.getID()
    TransactionRecord transaction = null

    Read the Transactions file      //named after the item ID in the format 'ID.txt'

    While the end of the file has not been reached
        Transaction = readTransaction(item, the line read)

    return transaction.getBalance()      //the balance of the latest transaction i.e. the the quantity in stock
}
```

## Process an entry or exit transaction

*A method of the TransactionRecord class*  
**Params:** none. All instance variables are attributes of a TransactionRecord object  
**Returns:** void

```
public void processTransaction() {

    int ID = item.getID()      //instance variable, an attribute of a TransactionRecord

    Read the Transactions file      //named after the item ID in the format 'ID.txt'
    Create a temporary file      //ID_Tmp.txt
```

```
int transactionID = 0
TransactionRecord previousTransaction = null

While the end of the file has not been reached
    previousTransaction = readTransaction(the line read)
    Write the line read from the Transactions file to the temporary file
    transactionID++

if previousTransaction == null
    set the balance to be the transaction's quantity
else
    if the transaction is an item entry
        set the balance to be the balance of previousTransaction + the balance of this transaction
    else          //the transaction is an item exit
        set the balance to be the balance of previousTransaction - the balance of this transaction

set the ID of the transaction to be transactionID

write the transaction to the end of the temporary file

Delete the transactions file
Rename the temporary file to 'ID_tmp.txt'

}
```

## OPERATIONS OF THE LINKED LIST OF TRANSACTION RECORDS

### Add a TransactionNode to the head of the list

**Params:**

- node: the TransactionNode to add

**Returns:** void

```
public void addToHead(TransactionNode node) {  
    if head == null //the list is empty  
        setHead(node)  
    else  
        node.setNext(head)  
        head.setPrev(node)  
        setHead(node)  
}
```

## Search for TransactionRecords between specified dates

Searches the list for transactions between specified dates by making a recursive call to the method searchTransactions(TransactionNode current, GregorianCalendar start, GregorianCalendar end, ArrayList searchResults)

**Params:**

- start: the lower boundary of the date range
- end: the upper boundary of the date range

**Returns:**

- an ArrayList of transactions between the specified dates

```
public ArrayList searchTransactions(GregorianCalendar start, GregorianCalendar end) {  
    return searchTransactions(head, start, end, new ArrayList())  
}
```

Method is called recursively to search the list for transactions between specified dates

**Params:**

- current: a pointer to the current node
- start: the lower boundary of the date range
- end: the upper boundary of the date range
- searchResults: an ArrayList of transactions between the specified dates

**Returns:**

- an ArrayList of transactions between the specified dates

```
public ArrayList searchTransactions(TransactionNode current, GregorianCalendar start,  
    GregorianCalendar end, ArrayList searchResults) {
```

```
if current == null
    if searchResults is empty
        return null
    else
        return searchResults

else if the date of current is greater or equal to start and less than or equal to end
    add current to searchResults
    return searchTransactions(current.getNext(), start, end, searchResults)
else if the date of current is greater than end
    //stop searching, past end date
    searchResults.trimToSize()
    return searchResults
else
    return searchTransactions(current.getNext(), start, end, searchResults)
}
```

## Delete a TransactionNode

Params:

- node: the node to delete

Returns: void

```
public void deleteTransaction(TransactionNode node) {

    if node.getPrev() is null          //it is the head of the tail
        setHead(node.getNext())       //set the next node to be the head
        head.setPrev(null)
        node.setNext(null)
        node.setPrev(null)
    else if node.getNext() is null     //it is the tail of the list
        node.getPrev().setNext(null)
        node.setPrev(null)
    else
}
```

```
node.getPrev().setNext(node.getNext())
node.setPrev(null)
node.setNext(null)

}
```

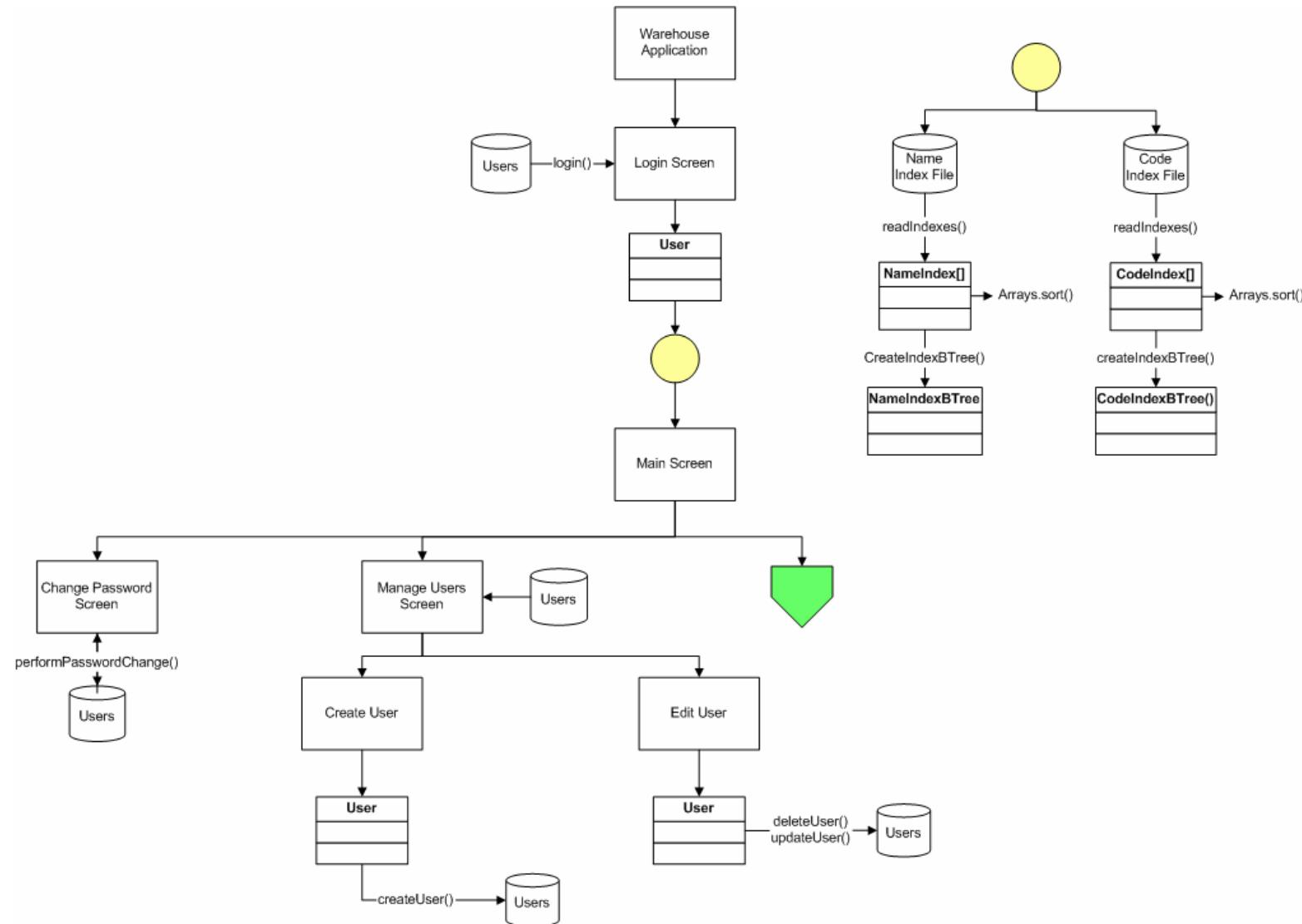
---

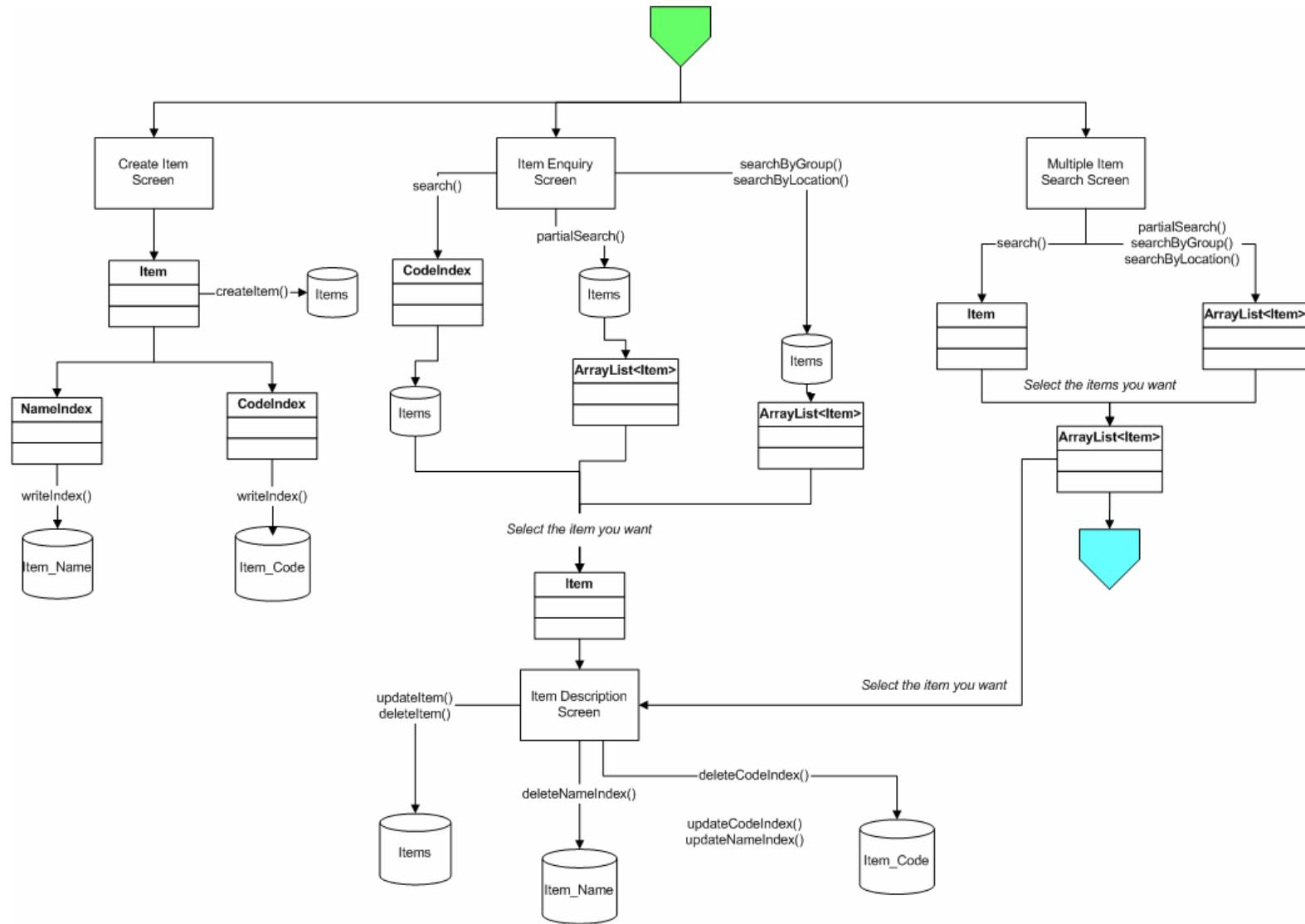
## B3 – Modular Design

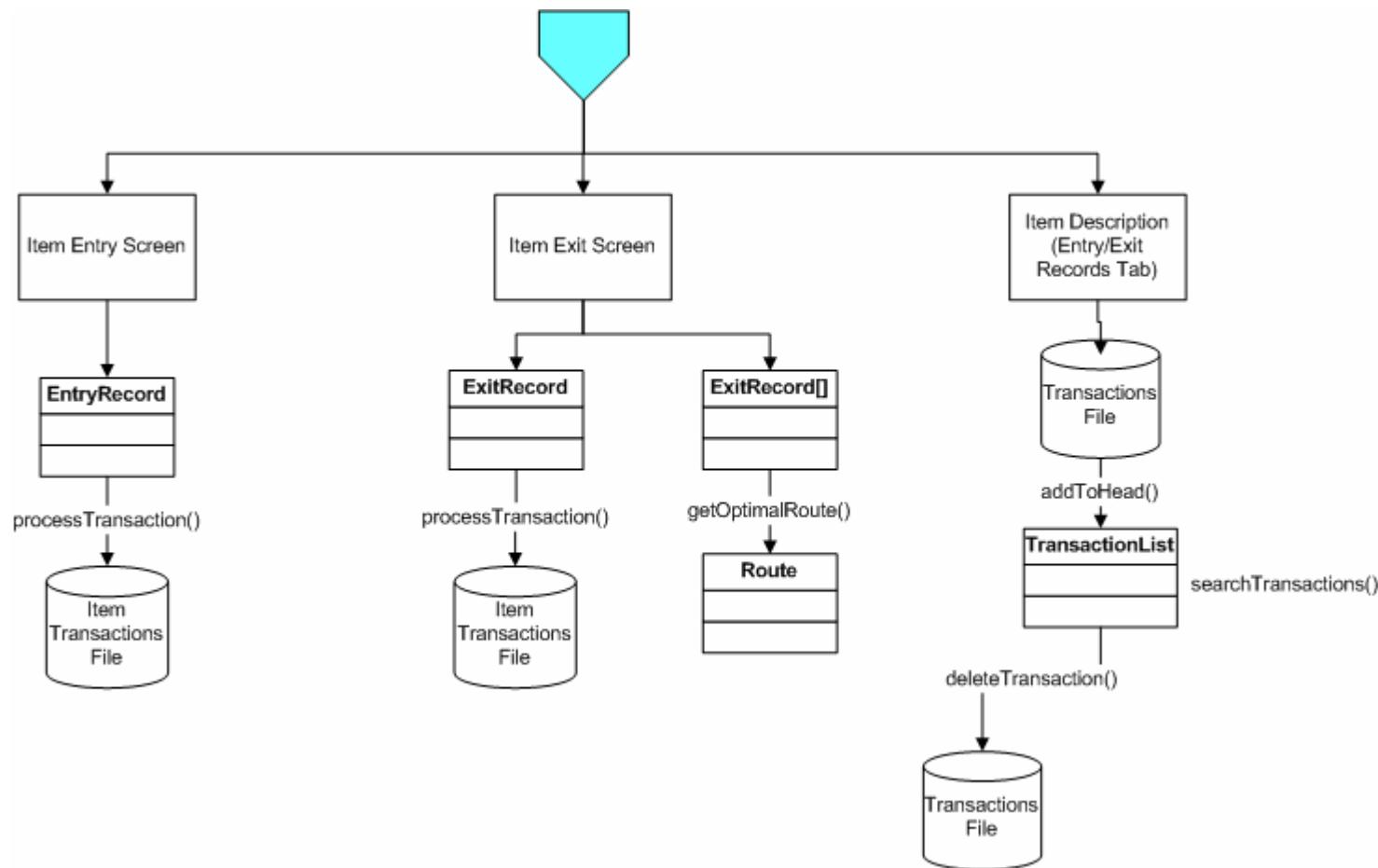
---

This section shows how all the components of the application, including the data structures, algorithms and prototypes fit together. It is intended as a concise summary of the detailed design phase.

Given the scale of this application, some parts have had to be referenced and continued separately so the diagrams fit the pages.







## Section C

---

# The Program

---

## C1 – Program Listing

---

### MASTERY ACHIEVED

Please refer to the specified location of the mastery factor for its explanation.  
All mastery in the code listing is denoted by the Javadoc identifier @mastery

Mastery Factor	Location	Page Number
1. Adding data to an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method	Achieved in the method <i>writeNewItem(Item item)</i> in line 547 of the class <i>CreateNewItem</i>	134
2. Deleting data from an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method.	Achieved in the method <i>deleteItem(Item item)</i> in line 850 of the class <i>ItemDescription</i>	208
3. Searching for specified data in a file.	Achieved in the method <i>login(String username, String password)</i> in line 177 of the class <i>LoginScreen</i>	280
4. Recursion	Achieved in the method <i>insertIndex(IndexTNode current, IndexTNode index)</i> in line 106 of the class <i>IndexBTree</i>	161
8. Encapsulation	Achieved in the class <i>Item</i>	173
9. Parsing a text file or other data stream	Achieved in the method <i>getGroups()</i> in line 147 of the class <i>MainScreen</i>	287
11. The use of any five standard level mastery factors	2. User-defined objects Achieved in the class <i>Item</i> 3. Objects as data records Achieved in the class <i>Item</i> 4. Simple selection	2. 173 3. 173 4. 292 6. 292

	Achieved in line 331 of the class <i>MainScreen</i>  6. Loops Achieved in line 325 of the class <i>MainScreen</i>  10. User-defined methods with appropriate return values Achieved in line 314 of the class <i>MainScreen</i>	10. 291
12-15. Implementation of ADTs	Achieved in the class <i>IndexBTree</i>	159
16. Use of additional libraries	Achieved in the class <i>OptimalRoute</i>	342
17. Inserting data into an ordered sequential file without reading the entire file into RAM.	Achieved in the method <i>writeNewUser(User user)</i> in line 244 of the class <i>NewUser</i>	336
18. Deleting data from a sequential file without reading the entire file into RAM.	Achieved in the method <i>deleteCodeIndex(Item item)</i> in line 993 of the class <i>ItemDescription</i>	212

```
1 import java.io.File;
2
3 /**
4 * -----
5 * Warehouse Application
6 * Class stores all global constants of the application, including file locations
7 * @author Alvaro Morales
8 * @date 03/06/2010
9 * @school Markham College
10 * @IDE Eclipse SDK
11 * @computer IBM ThinkPad R52
12 * -----
13 */
14 public interface ApplicationConstants {
15
16     //Location of program icons
17     public static final String CREATE_ITEM_ICON = "Icons" + File.separator + "CreateItemIcon.png";
18     public static final String ITEM_ENQUIRY_ICON = "Icons" + File.separator + "ItemEnquiryIcon.png";
19     public static final String ITEM_ENTRY_ICON = "Icons" + File.separator + "ItemEntryIcon.png";
20     public static final String ITEM_EXIT_ICON = "Icons" + File.separator + "ItemExitIcon.png";
21     public static final String WAREHOUSE_LOGO = "Icons" + File.separator + "WarehouseLogo.png";
22     public static final String USERS_ICON = "Icons" + File.separator + "ManageUsers.png";
23     public static final String PASSWORD_ICON = "Icons" + File.separator + "PasswordIcon.png";
24     public static final String NO_ITEM_IMAGE = "Files" + File.separator + "Item_Images"
25             + File.separator + "NoImage.jpg";
26
27     //Location of database files
28     public static final String USERS_FILE = "Files" + File.separator + "Users.txt";
29     public static final String ITEMS_FILE = "Files" + File.separator + "Items.dat";
30     public static final String NAME_INDEX_FILE = "Files" + File.separator + "Item_Name.txt";
31     public static final String CODE_INDEX_FILE = "Files" + File.separator + "Item_Code.txt";
32
33     //Location of info files
34     public static final String GROUPS_FILE = "Files" + File.separator + "Groups.txt";
35     public static final String UM_FILE = "Files" + File.separator + "UMs.txt";
36
```

```
37 //Location of temporary files
38 public static final String USERS_TMP_FILE = "Files" + File.separator + "Users_tmp.txt";
39 public static final String TMP_CODE_INDEX_FILE = "Files" + File.separator + "Item_Code_Tmp.txt";
40 public static final String TMP_NAME_INDEX_FILE = "Files" + File.separator + "Item_Name_Tmp.txt";
41
42 //Location of directories
43 public static final String IMAGES_FOLDER = "Images" + File.separator;
44 public static final String ITEM_IMAGES_FOLDER = "Files" + File.separator + "Item_Images"
45     + File.separator;
46 public static final String TRANSACTIONS_FOLDER = "Files" + File.separator + "Transactions"
47     + File.separator;
48
49 //Record lengths
50 public static final int ITEMS_FILE_RECORD_LENGTH = 869;
51
52 //Transaction Types
53 public static final int ITEM_ENTRY = 0;
54 public static final int ITEM_EXIT = 1;
55
56 //Transaction start date (when transactions start to be recorded)
57 public static final int TRANSACTION_YEAR_START = 2000;
58
59 }
```

```
1 import java.awt.event.*;
2 import java.io.*;
3 import java.util.*;
4 import javax.swing.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * Constructs the Account Settings screen and handles the operations to change a user's password
10 * @author Alvaro Morales
11 * @date 10/06/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class AccountSettings extends javax.swing.JFrame {
18
19 /**
20 * GUI components generated by Jigloo
21 */
22 private JLabel lblAccountSettings;
23 private JButton btnChangePassword;
24 private JPasswordField txtConfirmPassword;
25 private JPasswordField txtNewPassword;
26 private JPasswordField txtCurrentPassword;
27 private JLabel lblConfirm;
28 private JLabel lblNewPassword;
29 private JLabel lblCurrentPassword;
30
31 /**
32 * The user that logged in
33 */
34 private User user;
35
36 /**
```

```
37     * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
38     */
39 {
40     //Set Look & Feel
41     try {
42         javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
43     } catch(Exception e) {
44         e.printStackTrace();
45     }
46 }
47
48 /**
49 * Constructs a new AccountSettings object
50 * @param user - the user that logged in
51 */
52 public AccountSettings(User user) {
53     super();
54     this.user = user;
55     initGUI();
56 }
57
58 /**
59 * Method generated by Jigloo to initialize GUI components
60 */
61 private void initGUI() {
62     try {
63         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
64         getContentPane().setLayout(null);
65         this.setTitle("Account Settings");
66         this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
67             ApplicationConstants.PASSWORD_ICON)).getImage()
68     );
69     {
70         lblAccountSettings = new JLabel();
71         getContentPane().add(lblAccountSettings);
72         lblAccountSettings.setText("Change Password");
```

```
73     lblAccountSettings.setBounds(12, 12, 131, 16);
74     lblAccountSettings.setFont(new java.awt.Font("Dialog",1,14));
75 }
76 {
77     lblCurrentPassword = new JLabel();
78     getContentPane().add(lblCurrentPassword);
79     lblCurrentPassword.setText("Current Password:");
80     lblCurrentPassword.setBounds(44, 48, 107, 16);
81 }
82 {
83     lblNewPassword = new JLabel();
84     getContentPane().add(lblNewPassword);
85     lblNewPassword.setText("New Password:");
86     lblNewPassword.setBounds(62, 86, 89, 16);
87 }
88 {
89     lblConfirm = new JLabel();
90     getContentPane().add(lblConfirm);
91     lblConfirm.setText("Confirm Password:");
92     lblConfirm.setBounds(40, 122, 111, 16);
93 }
94 {
95     txtCurrentPassword = new JPasswordField();
96     getContentPane().add(txtCurrentPassword);
97     txtCurrentPassword.setBounds(163, 46, 156, 20);
98 }
99 {
100    txtNewPassword = new JPasswordField();
101    getContentPane().add(txtNewPassword);
102    txtNewPassword.setBounds(163, 84, 156, 20);
103 }
104 {
105    txtConfirmPassword = new JPasswordField();
106    getContentPane().add(txtConfirmPassword);
107    txtConfirmPassword.setBounds(163, 120, 156, 20);
108 }
```

```
109     {
110         btnChangePassword = new JButton();
111         getContentPane().add(btnChangePassword);
112         btnChangePassword.setText("Change");
113         btnChangePassword.setBounds(138, 155, 77, 26);
114         btnChangePassword.addActionListener(new ActionListener() {
115             public void actionPerformed (ActionEvent e){
116                 changePassword(
117                     new String(txtCurrentPassword.getPassword()),
118                     new String(txtNewPassword.getPassword()),
119                     new String(txtConfirmPassword.getPassword())
120                 );
121             }
122         });
123     }
124     pack();
125     this.setSize(351, 219);
126 } catch (Exception e) {
127     e.printStackTrace();
128 }
129 }
130
131 /**
132 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
133 */
134
135 /**
136 * Checks if the information entered in the JPasswordFields is correct.
137 * If so, it calls the performChange() method to change a user's password.
138 * @param currentPassword - the user's password
139 * @param newPassword - the new password
140 * @param confirmPassword - a confirmation of the new password (must be exactly equal to newPassword)
141 */
142 public void changePassword(String currentPassword, String newPassword, String confirmPassword){
143
144     try{
```

```
145     if(user.getPassword().equals(currentPassword) && (newPassword.equals(confirmPassword))){  
146         performChange(newPassword);  
147         JOptionPane.showMessageDialog(this, "Password successfully changed",  
148             "Information", JOptionPane.INFORMATION_MESSAGE);  
149         this.dispose();  
150     } else {  
151         JOptionPane.showMessageDialog(this, "Some fields are missing or the \ninformation " +  
152             "entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);  
153         txtCurrentPassword.setText("");  
154         txtNewPassword.setText("");  
155         txtConfirmPassword.setText("");  
156     }  
157  
158 } catch (Exception e) {  
159     JOptionPane.showMessageDialog(this, "The password has not been changed. \n" +  
160         "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);  
161     this.dispose();  
162 }  
163 }  
164  
165 /**  
 * Changes the password of a user in the file  
 * @param newPassword - the new password  
 */  
166 public void performChange(String newPassword) throws IOException {  
167     try{  
168         this.user.setPassword(newPassword);  
169  
170         File file = new File(ApplicationConstants.USERS_FILE);  
171         File tmpFile = new File(ApplicationConstants.USERS_TMP_FILE);      //Creates the temporary file  
172         FileReader reader = new FileReader(file);  
173         BufferedReader buff = new BufferedReader(reader);  
174         FileWriter writer = new FileWriter(tmpFile);  
175         BufferedWriter buffwriter = new BufferedWriter(writer);  
176     }  
177 }
```

```
181
182     boolean eof = false;           //stores if the end of the file (eof) has been reached
183
184     while(!eof){
185         String line = buff.readLine();
186         if(line == null){
187             eof = true;           //the end of the file has been reached
188         } else {
189             User userInFile = readUser(line);
190
191             //The file is already ordered, so no further validation is necessary
192             if(userInFile.getUsername().equals(this.user.getUsername())){
193                 buffwriter.write(user.toString());
194                 buffwriter.newLine();
195             } else {
196                 buffwriter.write(userInFile.toString());
197                 buffwriter.newLine();
198             }
199
200         }
201     }
202
203     buffwriter.close();
204     buff.close();
205     reader.close();
206     writer.close();
207
208     //Rename temporary file to old file
209     file.delete();
210     tmpFile.renameTo(file);
211
212 } catch (Exception e){
213
214 }
215
216 }
```

```
217
218 /**
219 * Creates a User object from a tokenized String input
220 * @param line - a line read from the Users file
221 * @return - a User object
222 */
223 public User readUser(String line){
224     StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
225
226     if (tokenizer.countTokens() == 7){          //preliminary error checking
227         String username = tokenizer.nextToken();
228         String password = tokenizer.nextToken();
229         String fullName = tokenizer.nextToken();
230         boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
231         boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
232         boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
233         boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
234
235         return new User(username, password, fullName, admin, entry, exit, userEnabled);
236     }
237
238     return null;
239
240 }
241
242 /**
243 * Converts an int to a boolean
244 * @param 0 if false, 1 if true
245 * @return true if param is 1, else false
246 */
247 public boolean intToBoolean(int i){
248     return (i==1);
249 }
250
251 }
```

```
1 import java.awt.BorderLayout;
2 import java.util.*;
3 import javax.swing.*;
4
5 /**
6 * -----
7 * Warehouse Application
8 * Class stores methods to change the screen of MainScreen's content pane
9 * @author Alvaro Morales
10 * @date 13/07/2010
11 * @school Markham College
12 * @IDE Eclipse SDK
13 * @computer IBM ThinkPad R52
14 * -----
15 */
16 public class ChangeScreen {
17
18     /**
19      * Changes the content pane of MainScreen to a blank panel
20      */
21     public static void setBlankScreen(User user){
22         JPanel pnlBlank = new JPanel();
23         pnlBlank.setSize(800, 500);
24         pnlBlank.setBackground(new java.awt.Color(192,192,192));
25         MainScreen.contentPane.removeAll();
26         MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
27         MainScreen.contentPane.add(pnlBlank, BorderLayout.CENTER);
28         MainScreen.contentPane.validate();
29     }
30
31     /**
32      * Changes the content pane of MainScreen to the Create New Item screen
33      * @param currentPanel - the current panel added to the content pane of MainScreen
34      * @param user - the user that logged in
35      */
```

```
36 public static void setCreateNewItemScreen(User user) {
37     CreateNewItem create = new CreateNewItem(user);
38     create.setSize(800, 500);
39     MainScreen.contentPane.removeAll();
40     MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
41     MainScreen.contentPane.add(create, BorderLayout.CENTER);
42     MainScreen.contentPane.validate();
43 }
44
45 /**
46 * Changes the content pane of MainScreen to the Manage Users screen
47 * @param currentPanel - the current panel added to the content pane of MainScreen
48 */
49 public static void setManageUsersScreen(User user) {
50     ManageUsers users = new ManageUsers();
51     users.setSize(800, 500);
52     MainScreen.contentPane.removeAll();
53     MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
54     MainScreen.contentPane.add(users, BorderLayout.CENTER);
55     MainScreen.contentPane.validate();
56 }
57
58 /**
59 * Changes the content pane of MainScreen to the Item Search screen
60 * @param currentPanel - the current panel added to the content pane of MainScreen
61 * @param user - the user that logged in
62 */
63 public static void setItemSearchScreen(User user) {
64     ItemSearch search = new ItemSearch(user);
65     search.setSize(800, 500);
66     MainScreen.contentPane.removeAll();
67     MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
68     MainScreen.contentPane.add(search, BorderLayout.CENTER);
69     MainScreen.contentPane.validate();
70 }
71 }
```

```
72  /**
73   * Changes the content pane of MainScreen to the Multiple Item Search screen
74   * @param currentPanel - the current panel added to the content pane of MainScreen
75   * @param user - the user that logged in
76   * @param transactionType - the type of transaction (Item entry or exit)
77   */
78  public static void setMultipleItemSearchScreen(User user, int transactionType) {
79      MultipleItemSearch search = new MultipleItemSearch(user, transactionType);
80      search.setSize(800, 500);
81      MainScreen.contentPane.removeAll();
82      MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
83      MainScreen.contentPane.add(search, BorderLayout.CENTER);
84      MainScreen.contentPane.validate();
85  }
86
87  /**
88   * Changes the content pane of MainScreen to the Optimal Route screen
89   * @param user - the user that logged in
90   * @param itemsToWithdraw - an ArrayList of items to withdraw from the warehouse
91   */
92  public static void setOptimalRouteScreen(User user, ArrayList itemsToWithdraw) {
93      OptimalRoute route = new OptimalRoute(itemsToWithdraw, user);
94      route.setSize(800, 500);
95      MainScreen.contentPane.removeAll();
96      MainScreen.contentPane.add(new WarehouseToolbar(user), BorderLayout.NORTH);
97      MainScreen.contentPane.add(route, BorderLayout.CENTER);
98      MainScreen.contentPane.validate();
99  }
100 }
```

```
1 import javax.swing.border.BevelBorder;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import java.util.*;
5 import java.io.*;
6 import javax.imageio.*;
7 import java.awt.*;
8
9 /**
10 * -----
11 * Warehouse Application
12 * The Create New Item screen to create a new item
13 * @author Alvaro Morales
14 * @date 02/07/2010
15 * @school Markham College
16 * @IDE Eclipse SDK
17 * @computer IBM ThinkPad R52
18 * -----
19 */
20 public class CreateNewItem extends javax.swing.JPanel {
21
22     /**
23      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
24      */
25
26     {
27         //Set Look & Feel
28         try {
29             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     /**

```

```
36     * GUI components generated by Jigloo
37     */
38
39     private JLabel lblCreateNewItem;
40     private JLabel lblRequiredFields;
41     private JLabel lblDescription;
42     private JButton btnBrowse;
43     private JButton btnCreate;
44     private JLabel lblRow;
45     private JLabel lblColumn;
46     private JLabel lblAisle;
47     private JLabel lblLocation;
48     private JTextField txtDate;
49     private JLabel lblDate;
50     private JTextField txtCreatedBy;
51     private JLabel lblCreatedBy;
52     private JLabel lblImage;
53     private JLabel lblItemImage;
54     private JTextArea txtDescription;
55     private JTextArea txtName;
56     private JLabel lblCodeInfo;
57     private JLabel lblCode;
58     private JTextField txtCode;
59     private JButton btnClearImage;
60     private JLabel lblImageInfo;
61     private JComboBox cmbRow;
62     private JComboBox cmbColumn;
63     private JComboBox cmbAisle;
64     private JLabel lblDescriptionInfo2;
65     private JLabel lblDescriptionInfo;
66     private JLabel lblNameInfo2;
67     private JLabel lblNameInfo;
68     private JComboBox cmbUM;
69     private JComboBox cmbGroup;
70     private JLabel lblUM;
71     private JLabel lblFamily;
```

```
72     private JLabel lblName;
73
74     /**
75      * The user that logged in
76      */
77     private User user;
78
79     /**
80      * The current date
81      */
82     private GregorianCalendar currentDate;
83
84     /**
85      * The item's image
86      */
87     private File imageFile;
88
89     /**
90      * Creates a new CreateNewItem object
91      * @param user - the user that logged in
92      */
93     public CreateNewItem(User user) {
94         super();
95         this.user = user;
96         initGUI();
97     }
98
99     /**
100      * Methods generated by Jigloo to initialize GUI components
101      */
102     private void initGUI() {
103         try {
104             this.setLayout(null);
105             this.setPreferredSize(new java.awt.Dimension(800, 500));
106             this.setSize(800, 500);
107         }
```

```
108     lblCreateNewItem = new JLabel();
109     this.add(lblCreateNewItem);
110     lblCreateNewItem.setText("Create a New Item");
111     lblCreateNewItem.setFont(new java.awt.Font("Dialog",1,14));
112     lblCreateNewItem.setBounds(12, 12, 148, 14);
113 }
114 {
115     lblRequiredFields = new JLabel();
116     this.add(lblRequiredFields);
117     lblRequiredFields.setText("Fields marked with an * are required");
118     lblRequiredFields.setBounds(12, 29, 210, 16);
119     lblRequiredFields.setFont(new java.awt.Font("Dialog",0,10));
120 }
121 {
122     lblName = new JLabel();
123     this.add(lblName);
124     lblName.setText("Name* :");
125     lblName.setBounds(65, 94, 44, 16);
126 }
127 {
128     lblFamily = new JLabel();
129     this.add(lblFamily);
130     lblFamily.setText("Group* :");
131     lblFamily.setBounds(62, 198, 46, 13);
132 }
133 {
134     lblUM = new JLabel();
135     this.add(lblUM);
136     lblUM.setText("U.M.* :");
137     lblUM.setBounds(74, 238, 35, 10);
138 }
139 {
140     lblDescription = new JLabel();
141     this.add(lblDescription);
142     lblDescription.setText("Description:");
143     lblDescription.setBounds(41, 269, 68, 18);
```

```
144 }
145 {
146     btnBrowse = new JButton();
147     this.add(btnBrowse);
148     btnBrowse.setText("Browse...");
149     btnBrowse.setBounds(533, 287, 89, 26);
150     btnBrowse.addActionListener(new ActionListener() {
151         public void actionPerformed (ActionEvent e){
152             getAndDisplayImage();
153         }
154     });
155 }
156 {
157     ComboBoxModel cmbFamilyModel =
158         new DefaultComboBoxModel(MainScreen.groups);
159     cmbGroup = new JComboBox();
160     this.add(cmbGroup);
161     cmbGroup.setModel(cmbFamilyModel);
162     cmbGroup.setBounds(121, 193, 191, 22);
163     cmbGroup.setSize(191, 25);
164 }
165 {
166     ComboBoxModel cmbUMModel =
167         new DefaultComboBoxModel(MainScreen.umArray);
168     cmbUM = new JComboBox();
169     this.add(cmbUM);
170     cmbUM.setModel(cmbUMModel);
171     cmbUM.setBounds(121, 232, 193, 25);
172 }
173 {
174     txtDescription = new JTextArea();
175     this.add(txtDescription);
176     txtDescription.setBounds(121, 270, 350, 85);
177     txtDescription.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
178     txtDescription.setLineWrap(true);
179     txtDescription.setDocument(new TextFieldLimit(200));
```

```
180 }
181 {
182     lblItemImage = new JLabel();
183     this.add(lblItemImage);
184     lblItemImage.setBounds(533, 85, 224, 168);
185     lblItemImage.setBorder(BorderFactory.createEtchedBorder(BevelBorder.LOWERED));
186 }
187 {
188     lblImage = new JLabel();
189     this.add(lblImage);
190     lblImage.setText("Image:");
191     lblImage.setBounds(533, 68, 48, 16);
192 }
193 {
194     lblCreatedBy = new JLabel();
195     this.add(lblCreatedBy);
196     lblCreatedBy.setText("Created By:");
197     lblCreatedBy.setBounds(42, 374, 69, 15);
198 }
199 {
200     txtCreatedBy = new JTextField();
201     this.add(txtCreatedBy);
202     txtCreatedBy.setBounds(121, 368, 139, 28);
203     txtCreatedBy.setEditable(false);
204     txtCreatedBy.setSize(139, 25);
205 }
206 {
207     lblDate = new JLabel();
208     this.add(lblDate);
209     lblDate.setText("Date:");
210     lblDate.setBounds(272, 374, 30, 14);
211 }
212 {
213     txtDate = new JTextField();
214     this.add(txtDate);
215     txtDate.setBounds(313, 368, 158, 27);
```

```
216         txtDate.setEditable(false);
217         txtDate.setSize(158, 25);
218     }
219     {
220         lblLocation = new JLabel();
221         this.add(lblLocation);
222         lblLocation.setText("Location");
223         lblLocation.setBounds(45, 409, 64, 7);
224         lblLocation.setFont(new java.awt.Font("Dialog",1,14));
225         lblLocation.setSize(64, 16);
226     }
227     {
228         lblAisle = new JLabel();
229         this.add(lblAisle);
230         lblAisle.setText("Aisle* :");
231         lblAisle.setBounds(71, 437, 40, 4);
232         lblAisle.setSize(40, 10);
233     }
234     {
235         lblColumn = new JLabel();
236         lblColumn.setText("Column* :");
237         lblColumn.setBounds(334, 437, 55, 12);
238         this.add(lblColumn);
239     }
240     {
241         lblRow = new JLabel();
242         lblRow.setText("Row* :");
243         lblRow.setBounds(207, 436, 36, 10);
244         this.add(lblRow);
245     }
246     {
247         btnCreate = new JButton();
248         btnCreate.setText("Create Item");
249         btnCreate.setBounds(640, 441, 100, 26);
```

```
252     btnCreate.addActionListener(new ActionListener() {
253         public void actionPerformed (ActionEvent e){
254             if(isCorrectInput()){
255                 performItemCreation();
256             } else {
257                 displayInputError();
258             }
259         }
260     });
261     this.add(btnCreate);
262     this.add(getTxtName());
263     this.add(getLblNameInfo());
264     this.add(getLblNameInfo2());
265     this.add(getLblDescriptionInfo());
266     this.add(getLblDescriptionInfo2());
267     this.add(getCmbAisle());
268     this.add(getCmbColumn());
269     this.add(getCmbRow());
270     this.add(getLblImageInfo());
271     this.add(getBtnClearImage());
272     this.add(getTxtCode());
273     this.add(getLblCode());
274     this.add(getLblCodeInfo());
275
276     }
277     {
278         fillDisabledFields();
279     }
280 } catch (Exception e) {
281     e.printStackTrace();
282 }
283
284 }
285
286 private JTextArea getTxtName() {
287     if(txtName == null) {
```

```
288     txtName = new JTextArea();
289     txtName.setBounds(121, 95, 350, 84);
290     txtName.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
291     txtName.setLineWrap(true);
292     txtName.setDocument(new TextFieldLimit(200));
293 }
294 return txtName;
295 }
296
297 private JLabel getLblNameInfo() {
298     if(lblNameInfo == null) {
299         lblNameInfo = new JLabel();
300         lblNameInfo.setText("No more than");
301         lblNameInfo.setBounds(45, 107, 67, 16);
302         lblNameInfo.setFont(new java.awt.Font("Dialog",0,10));
303     }
304     return lblNameInfo;
305 }
306
307 private JLabel getLblNameInfo2() {
308     if(lblNameInfo2 == null) {
309         lblNameInfo2 = new JLabel();
310         lblNameInfo2.setText("200 characters");
311         lblNameInfo2.setBounds(39, 122, 72, 15);
312         lblNameInfo2.setFont(new java.awt.Font("Dialog",0,10));
313     }
314     return lblNameInfo2;
315 }
316
317 private JLabel getLblDescriptionInfo() {
318     if(lblDescriptionInfo == null) {
319         lblDescriptionInfo = new JLabel();
320         lblDescriptionInfo.setText("No more than");
321         lblDescriptionInfo.setBounds(45, 285, 76, 16);
322         lblDescriptionInfo.setFont(new java.awt.Font("Dialog",0,10));
323     }
}
```

```
324     return lblDescriptionInfo;
325 }
326
327 private JLabel getLblDescriptionInfo2() {
328     if(lblDescriptionInfo2 == null) {
329         lblDescriptionInfo2 = new JLabel();
330         lblDescriptionInfo2.setText("200 characters");
331         lblDescriptionInfo2.setBounds(38, 301, 87, 15);
332         lblDescriptionInfo2.setFont(new java.awt.Font("Dialog",0,10));
333     }
334     return lblDescriptionInfo2;
335 }
336
337 private JComboBox getCmbAisle() {
338     if(cmbAisle == null) {
339         ComboBoxModel cmbAisleModel = new DefaultComboBoxModel(MainScreen.locations.getAisles());
340         cmbAisle = new JComboBox();
341         cmbAisle.setModel(cmbAisleModel);
342         cmbAisle.setBounds(122, 430, 67, 24);
343         cmbAisle.setSize(67, 25);
344     }
345     return cmbAisle;
346 }
347
348 private JComboBox getCmbColumn() {
349     if(cmbColumn == null) {
350         ComboBoxModel cmbColumnModel = new DefaultComboBoxModel(MainScreen.locations.getColumns());
351         cmbColumn = new JComboBox();
352         cmbColumn.setModel(cmbColumnModel);
353         cmbColumn.setBounds(400, 431, 67, 25);
354     }
355     return cmbColumn;
356 }
357
358 private JComboBox getCmbRow() {
359     if(cmbRow == null) {
```

```
360     ComboBoxModel cmbRowModel = new DefaultComboBoxModel(MainScreen.locations.getRows());
361     cmbRow = new JComboBox();
362     cmbRow.setModel(cmbRowModel);
363     cmbRow.setBounds(255, 431, 67, 27);
364     cmbRow.setSize(67, 25);
365 }
366 return cmbRow;
367 }

368
369 private JLabel getLblImageInfo() {
370     if(lblImageInfo == null) {
371         lblImageInfo = new JLabel();
372         lblImageInfo.setText("Must be 320x240 pixels");
373         lblImageInfo.setBounds(533, 259, 134, 16);
374         lblImageInfo.setFont(new java.awt.Font("Dialog",0,10));
375     }
376     return lblImageInfo;
377 }
378

379 private JButton getBtnClearImage() {
380     if(btnClearImage == null) {
381         btnClearImage = new JButton();
382         btnClearImage.setText("Clear Image");
383         btnClearImage.setBounds(655, 287, 102, 26);
384         btnClearImage.setEnabled(false);
385         btnClearImage.addActionListener(new ActionListener() {
386             public void actionPerformed (ActionEvent e){
387                 clearImage();
388             }
389         });
390     }
391     return btnClearImage;
392 }
393
394 }
395 }
```

```
396     private JTextField getTxtCode() {
397         if(txtCode == null) {
398             txtCode = new JTextField();
399             txtCode.setBounds(121, 57, 191, 20);
400             txtCode.setSize(191, 25);
401             txtCode.setDocument(new TextFieldLimit(6));
402         }
403         return txtCode;
404     }
405
406     private JLabel getLblCode() {
407         if(lblCode == null) {
408             lblCode = new JLabel();
409             lblCode.setText("Code* :");
410             lblCode.setBounds(69, 57, 40, 17);
411         }
412         return lblCode;
413     }
414
415     private JLabel getLblCodeInfo() {
416         if(lblCodeInfo == null) {
417             lblCodeInfo = new JLabel();
418             lblCodeInfo.setText("Exactly 6 numbers");
419             lblCodeInfo.setBounds(21, 71, 94, 16);
420             lblCodeInfo.setFont(new java.awt.Font("Dialog",0,10));
421         }
422         return lblCodeInfo;
423     }
424
425     /**
426      * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
427     */
428
429     /**
430      * Fills the 'Created By' and 'Date' disabled fields when the screen renders
431     */
```

```
432 public void fillDisabledFields(){
433     txtCreatedBy.setText(user.getUsername());      //Fill 'Created By'
434
435     //Create Date and output in DD/MM/YYYY format
436     currentDate = new GregorianCalendar();
437     String date =
438         currentDate.get(Calendar.DAY_OF_MONTH) + "/" +
439         currentDate.get(Calendar.MONTH) + "/" +
440         currentDate.get(Calendar.YEAR);
441     txtDate.setText(date);
442 }
443
444 /**
445 * Informs the user that the input is invalid or incorrect
446 */
447 public void displayInputError(){
448     JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);
449 }
450
451 /**
452 * Informs the user that the item has been created
453 */
454 public void displaySuccessMessage(){
455     JOptionPane.showMessageDialog(this, "The item has been created.",
456         "Success", JOptionPane.INFORMATION_MESSAGE);
457     ChangeScreen.setBlankScreen(user);
458 }
459
460 /**
461 * Gets an image from the File that the user selected and displays it in the Product Image box
462 */
463 public void getAndDisplayImage(){
464     //Get and Display a File Chooser
465     JFileChooser fileChooser = new JFileChooser();
466     fileChooser.setAcceptAllFileFilterUsed(false);
467     fileChooser.setFileFilter(new JpgFilter());
```

```
468     fileChooser.setDialogTitle("Choose an item image");
469     int returnValue = fileChooser.showOpenDialog(this);
470
471     if (returnValue == JFileChooser.APPROVE_OPTION){
472         if(fileChooser.getSelectedFile() != null){
473             File file = fileChooser.getSelectedFile();
474             this.imageFile = file;
475             try{
476                 Image image = ImageIO.read(file);
477                 lblItemImage.setIcon(new ImageIcon(image));
478                 btnClearImage.setEnabled(true);
479
480             } catch (Exception e){
481                 JOptionPane.showMessageDialog(
482                     MainScreen.contentPane, "One or files cannot be accessed. \n"
483                     + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
484             }
485         }
486     }
487
488 }
489
490 /**
491 * Clears the Image from the product image box
492 */
493 public void clearImage(){
494     lblItemImage.setIcon(null);
495     btnClearImage.setEnabled(false);
496 }
497
498 /**
499 * Validates the user input
500 * @return true if input is valid, else false
501 */
502 public boolean isCorrectInput(){
```

```
504     if(!txtName.getText().equals("") || txtName.getText() != null
505         && txtCode.getText().length() == 6){
506         return true;
507     } else {
508         return false;
509     }
510 }
511 }
512 /**
513 * Creates a new Item from user input in the Create New Item screen
514 * @return a new Item from user input
515 */
516
517 public Item createItemFromInput(){
518     int ID = Integer.MAX_VALUE;           //Temporary value for ID; will be changed later
519     String code = txtCode.getText();
520     String name = txtName.getText();
521     Group group = ((Group)cmbGroup.getSelectedItem());
522     UM um = ((UM)cmbUM.getSelectedItem());
523     String description = txtDescription.getText();
524     String createdBy = txtCreatedBy.getText();
525     int day = currentDate.get(Calendar.DAY_OF_MONTH);
526     int month = currentDate.get(Calendar.MONTH);
527     int year = currentDate.get(Calendar.YEAR);
528     GregorianCalendar creationDate = new GregorianCalendar(year, month, day);
529     Location location = new Location(
530         (Integer)cmbAisle.getSelectedItem().byteValue(),
531         ((Integer)cmbColumn.getSelectedItem()).byteValue(),
532         ((String)cmbRow.getSelectedItem()).charAt(0)
533     );
534
535     Item item = new Item(ID, code, name, group, um, description, location, createdBy, creationDate);
536
537     return item;
538 }
539 }
```

```
540     /**
541      * Writes a new Item to the Items Random Access File
542      * @param item - the item to write to the file
543      * @mastery achieves HL mastery factor 1 by adding a new item to the Items Random Access File using seek and
544      * write methods
545      * by manipulating the file pointer using the seek method
546      */
547  public void writeNewItem(Item item) {
548
549      try{
550          RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
551
552          if(itemsFile.length() == 0){
553              initItemsFile(itemsFile);
554          }
555
556          int emptyRecordPos = 0;           //Position of the first empty record
557          boolean isFull = false;          //Marks if the end of the file has been reached
558
559          itemsFile.seek(0);
560
561          while(itemsFile.readInt() != -999 && !isFull){
562              emptyRecordPos = emptyRecordPos+1;
563              if(emptyRecordPos >= (
564                  itemsFile.length() / ApplicationConstants.ITEMS_FILE_RECORD_LENGTH)) {
565                  isFull = true;
566              } else {
567                  itemsFile.seek(emptyRecordPos * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
568              }
569          }
570
571          if(isFull){
572              growItemsFile(itemsFile);
573          }
574
575          if(emptyRecordPos != 0){
```

```
576     itemsFile.seek((emptyRecordPos-1) * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
577     int prevID = itemsFile.readInt();
578     itemsFile.seek(emptyRecordPos*ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
579
580     //Write the item
581     item.setID(prevID + 1);
582     item.writeItemToFile(itemsFile);
583 } else {
584     item.setID(0);
585     itemsFile.seek(0);
586     item.writeItemToFile(itemsFile);
587 }
588
589 if(imageFile!=null){
590     File newImageFile = copyFile(imageFile, ApplicationConstants.ITEM_IMAGES_FOLDER);
591     newImageFile.renameTo(new File(ApplicationConstants.ITEM_IMAGES_FOLDER + item.getID() + ".jpg"));
592 }
593
594 Index codeIndex = new Index(item.getID(), item.getCode());
595 Index nameIndex = new Index(item.getID(), item.getName());
596
597 writeIndex(codeIndex, 0);
598 writeIndex(nameIndex, 1);
599
600 MainScreen.codeIndexTree.insertIndex(new IndexTNode(codeIndex));
601 MainScreen.nameIndexTree.insertIndex(new IndexTNode(nameIndex));
602
603 //Create a Transactions file
604 File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + item.getID() + ".txt");
605 file.createNewFile();
606
607     itemsFile.close();
608 } catch (Exception e){
609
610 }
611
```

```
612     }
613
614     /**
615      * Initialize the Items Random Access File to 20 empty records.
616      * @param itemsFile - the Items Random Access File
617      */
618     public void initItemsFile(RandomAccessFile itemsFile){
619         int maxRecords = 10;
620
621         String emptyNameDescription = "";
622         while(emptyNameDescription.length()!= 200){
623             emptyNameDescription = emptyNameDescription + "X";
624         }
625
626         String emptyCode = "";
627         while(emptyCode.length()!= 6){
628             emptyCode = emptyCode + "X";
629         }
630
631         String emptyCreatedBy = "";
632         while(emptyCreatedBy.length()!= 20){
633             emptyCreatedBy = emptyCreatedBy + "X";
634         }
635
636         try{
637             for(int i=0;i<maxRecords;i++){
638                 itemsFile.seek(i * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
639                 writeEmptyRecord(itemsFile, emptyNameDescription, emptyCode, emptyCreatedBy);
640             }
641         } catch (Exception e){
642
643         }
644     }
645
646     /**
647      * Writes a new empty record to the Items Random Access File.
```

```
648     * An ID "-999" denotes that the record is empty.  
649     * @param itemsFile - the Items Random Access File  
650     * @param emptyNameDescription - an empty String for the Name or Description, 200 characters long  
651     * @param emptyCode - an empty String for the Code, 6 characters long  
652     * @param emptyCreatedBy - an empty String for the createdBy field, 20 characters long  
653     */  
654     public void writeEmptyRecord(RandomAccessFile itemsFile, String emptyNameDescription,  
655         String emptyCode, String emptyCreatedBy){  
656         try{  
657             itemsFile.writeInt(-999);  
658             itemsFile.writeUTF(emptyCode);  
659             itemsFile.writeUTF(emptyNameDescription);  
660             itemsFile.writeShort(-1);  
661             itemsFile.writeShort(-1);  
662             itemsFile.writeUTF(emptyNameDescription);  
663             itemsFile.writeByte(-1);  
664             itemsFile.writeByte(-1);  
665             itemsFile.writeByte(-1);  
666             itemsFile.writeChar('X');  
667             itemsFile.writeUTF(emptyCreatedBy);  
668             itemsFile.writeShort(-1);  
669             itemsFile.writeByte(-1);  
670             itemsFile.writeByte(-1);  
671         } catch (Exception e){  
672             }  
673         }  
674     }  
675  
676     /**  
677     * Grows the maximum number of records in the Items Random Access File by a factor of 1.5  
678     * @param itemsFile - the maximum number of records in the Items Random Access File  
679     */  
680     public void growItemsFile(RandomAccessFile itemsFile){  
681         try{  
682             long numberOfRecords = (itemsFile.length() / ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);  
683             long newNumberOfRecords = (long)(numberOfRecords * 1.5);
```

```
684     itemsFile.setLength(newNumberOfRecords * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
685     itemsFile.seek(numberOfRecords * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
686
687     String emptyNameDescription = "";
688     while(emptyNameDescription.length()!= 200){
689         emptyNameDescription = emptyNameDescription + "X";
690     }
691
692     String emptyCode = "";
693     while(emptyCode.length()!= 6){
694         emptyCode = emptyCode + "X";
695     }
696
697     String emptyCreatedBy = "";
698     while(emptyCreatedBy.length()!= 20){
699         emptyCreatedBy = emptyCreatedBy + "X";
700     }
701
702     for(long i = numberOfRecords;i<newNumberOfRecords;i++){
703         itemsFile.seek(i * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
704         writeEmptyRecord(itemsFile, emptyNameDescription, emptyCode, emptyCreatedBy);
705     }
706
707 } catch (Exception e){
708 }
709
710 }
711
712 /**
713 * Copies a file to a destination
714 * @author Vinod Singh, with modifications by Alvaro Morales
715 * @authorswebsite http://blog.vinodsingh.com/2009/06/copy-move-and-delete-files-using-java.html
716 * @date 12 June 2009
717 * @param file - the file to copy
718 * @param destination - the destination path of the file
```

```
720     * @return the file that has been copied
721     * @throws IOException if file accessing does not work
722     */
723     public File copyFile(File file, String destination) throws IOException {
724         FileChannel in = null;
725         FileChannel out = null;
726         try {
727             in = new FileInputStream(file).getChannel();
728             File outFile = new File(destination, file.getName());
729             out = new FileOutputStream(outFile).getChannel();
730             in.transferTo(0, in.size(), out);
731             return outFile;
732         } finally {
733             if (in != null)
734                 in.close();
735             if (out != null)
736                 out.close();
737         }
738     }
739
740
741 /**
742 * Creates an index by code or name of the item being created
743 * @param index - the Index by code object
744 * @param indexType - the type of Index:
745 * 0 if it is an index by code, 1 if it is an index by name.
746 */
747 public void writeIndex(Index index, int indexType){
748     if(indexType == 0){
749         try{
750             File file = new File(ApplicationConstants.CODE_INDEX_FILE);
751             File tmpFile = new File(ApplicationConstants.TMP_CODE_INDEX_FILE);
752             FileReader reader = new FileReader(file);
753             BufferedReader buff = new BufferedReader(reader);
754             FileWriter writer = new FileWriter(tmpFile);
755             BufferedWriter buffwriter = new BufferedWriter(writer);
```

```
756
757     boolean eof = false;           //stores if the end of the file (eof) has been reached
758
759     while(!eof){
760         String line = buff.readLine();
761         if(line == null){
762             eof = true;          //the end of the file has been reached
763         } else {
764             buffwriter.write(line);
765             buffwriter.newLine();
766
767         }
768     }
769
770     buffwriter.write(index.toString());
771
772     reader.close();
773     buff.close();
774     buffwriter.close();
775     writer.close();
776
777     //Rename temporary file to old file
778     file.delete();
779     tmpFile.renameTo(file);
780
781 } catch (Exception e){
782
783 }
784 } else {
785 try{
786     File file = new File(ApplicationConstants.NAME_INDEX_FILE);
787     File tmpFile = new File(ApplicationConstants.TMP_NAME_INDEX_FILE);
788     FileReader reader = new FileReader(file);
789     BufferedReader buff = new BufferedReader(reader);
790     FileWriter writer = new FileWriter(tmpFile);
791     BufferedWriter buffwriter = new BufferedWriter(writer);
```

```
792
793     boolean eof = false;           //stores if the end of the file (eof) has been reached
794
795     while(!eof){
796         String line = buff.readLine();
797         if(line == null){
798             eof = true;          //the end of the file has been reached
799         } else {
800             buffwriter.write(line);
801             buffwriter.newLine();
802         }
803     }
804
805     buffwriter.write(index.toString());
806
807     reader.close();
808     buff.close();
809     buffwriter.close();
810     writer.close();
811
812     //Rename temporary file to old file
813     file.delete();
814     tmpFile.renameTo(file);
815 } catch (Exception e){
816
817 }
818 }
819 }
820
821 /**
822 * Performs the item creation.
823 * Checks to see if the location set by the user is taken.
824 * If it is, it informs the user and asks to proceed with creation anyway
825 */
826 public void performItemCreation(){
827     Item newItem = createItemFromInput();
```

```
828     ArrayList itemsInLocation = Items.searchByLocation(newItem.getLocation());
829
830     if(itemsInLocation == null || itemsInLocation.isEmpty()){
831         writeNewItem(newItem);
832         displaySuccessMessage();
833     } else {
834         String items = "The following items are stored in the location " + newItem.getLocation().toString()
835         + ": \n";
836         for(int i=0;i<itemsInLocation.size();i++){
837             Item item = (Item)itemsInLocation.get(i);
838             String itemName = item.getName();
839             if(itemName.length()>40){
840                 itemName = itemName.substring(0, 19) + "...";
841             }
842             items = items + "\n" + (i+1) + ". " + itemName;
843         }
844
845         items = items + "\n" + "\n" + "Assign this location to the new item anyway?";
846
847         int userResponse = JOptionPane.showConfirmDialog(
848             this, items, "Location Taken",
849             JOptionPane.YES_NO_OPTION);
850         if(userResponse == JOptionPane.YES_OPTION){
851             writeNewItem(newItem);
852             displaySuccessMessage();
853         }
854     }
855 }
856 }
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.io.*;
4 import java.util.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * Constructs the EditUser screen to update a user
10 * @author Alvaro Morales
11 * @date 10/06/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class EditUser extends javax.swing.JFrame {
18
19     /**
20      * GUI components generated by Jigloo
21      */
22     private JLabel lblEditUser;
23     private JLabel lblFullName;
24     private JCheckBox chkAdmin;
25     private JCheckBox chkEnabled;
26     private JLabel lblStatus;
27     private JLabel lblPasswordValidation;
28     private JLabel lblPermissionsValidation;
29     private JLabel lblUsernameValidation;
30     private JSeparator spEdit;
31     private JButton btnSave;
32     private JCheckBox chkExit;
33     private JCheckBox chkEntry;
34     private JTextField txtFullName;
35     private JTextField txtPassword;
```

```
36     private JTextField txtUsername;
37     private JLabel lblPermissions;
38     private JLabel lblPassword;
39     private JLabel lblUsername;
40
41     /**
42      * The user is being edited
43      */
44     private User user;
45
46     /**
47      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
48      */
49
50     {
51         //Set Look & Feel
52         try {
53             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
54         } catch(Exception e) {
55             e.printStackTrace();
56         }
57     }
58
59
60     /**
61      * Creates a new EditUser object
62      * @param user - the user that is being edited
63      */
64     public EditUser(User user) {
65         super();
66         this.user = user;
67         initGUI();
68     }
69
70     /**
71      * Method generated by Jigloo to initialize GUI components
```

```
72     */
73     private void initGUI() {
74         try {
75             setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
76             this.setTitle("Edit User");
77             getContentPane().setLayout(null);
78         {
79             lblEditUser = new JLabel();
80             getContentPane().add(lblEditUser);
81             lblEditUser.setText("Edit User");
82             lblEditUser.setBounds(12, 12, 139, 16);
83             lblEditUser.setFont(new java.awt.Font("Dialog",1,14));
84         }
85         {
86             lblUsername = new JLabel();
87             getContentPane().add(lblUsername);
88             lblUsername.setText("Username:");
89             lblUsername.setBounds(51, 40, 62, 16);
90         }
91         {
92             lblPassword = new JLabel();
93             getContentPane().add(lblPassword);
94             lblPassword.setText("Password:");
95             lblPassword.setBounds(53, 76, 61, 16);
96         }
97         {
98             lblFullName = new JLabel();
99             getContentPane().add(lblFullName);
100            lblFullName.setText("Full Name:");
101            lblFullName.setBounds(56, 113, 58, 16);
102        }
103        {
104            lblPermissions = new JLabel();
105            getContentPane().add(lblPermissions);
106            lblPermissions.setText("Permissions:");
107            lblPermissions.setBounds(39, 143, 75, 16);
```

```
108
109
110     txtUsername = new JTextField();
111     txtUsername.setDocument(new TextFieldLimit(20));
112     getContentPane().add(txtUsername);
113     txtUsername.setBounds(127, 40, 250, 20);
114     txtUsername.setSize(200, 20);
115     txtUsername.setText(user.getUsername());
116     txtUsername.setEnabled(false);
117 }
118 {
119     txtPassword = new JTextField();
120     txtPassword.setDocument(new TextFieldLimit(30));
121     getContentPane().add(txtPassword);
122     txtPassword.setBounds(126, 75, 201, 20);
123     txtPassword.setText(user.getPassword());
124 }
125 {
126     txtFullName = new JTextField();
127     getContentPane().add(txtFullName);
128     txtFullName.setBounds(126, 112, 200, 20);
129     txtFullName.setText(user.getFullName());
130 }
131 {
132     chkAdmin = new JCheckBox();
133     getContentPane().add(chkAdmin);
134     chkAdmin.setText("Admin");
135     chkAdmin.setBounds(128, 160, 61, 24);
136     chkAdmin.setSelected(user.isAdmin());
137
138     chkAdmin.addActionListener(new ActionListener() {
139         public void actionPerformed (ActionEvent e){
140             if(chkAdmin.isSelected()){
141                 chkEntry.setEnabled(false);
142                 chkEntry.setSelected(true);
143                 chkExit.setEnabled(false);
```

```
144             chkExit.setSelected(true);
145         } else {
146             chkEntry.setEnabled(true);
147             chkEntry.setSelected(false);
148             chkExit.setEnabled(true);
149             chkExit.setSelected(false);
150         }
151     }
152 }
153 );
154 }
155 {
156     chkEntry = new JCheckBox();
157     getContentPane().add(chkEntry);
158     chkEntry.setText("Item Entry Operations");
159     chkEntry.setBounds(128, 185, 148, 24);
160     chkEntry.setSelected(user.isEntry());
161 }
162
163 {
164     chkExit = new JCheckBox();
165     getContentPane().add(chkExit);
166     chkExit.setText("Item Exit Operations");
167     chkExit.setBounds(128, 211, 140, 24);
168     chkExit.setSelected(user.isExit());
169 }
170
171 {
172     if(chkAdmin.isSelected()){
173         chkEntry.setEnabled(false);
174         chkExit.setEnabled(false);
175     }
176     {
177         btnSave = new JButton();
178         getContentPane().add(btnSave);
179         btnSave.setText("Save Changes");
```

```
180     btnSave.setBounds(128, 300, 122, 26);
181     btnSave.addActionListener(new ActionListener() {
182         public void actionPerformed (ActionEvent e){
183             editUser();
184         }
185     });
186 }
187 {
188     spEdit = new JSeparator();
189     getContentPane().add(spEdit);
190     spEdit.setBounds(12, 285, 342, 10);
191 }
192 {
193     lblUsernameValidation = new JLabel();
194     getContentPane().add(lblUsernameValidation);
195     lblUsernameValidation.setText("Max. 20 characters");
196     lblUsernameValidation.setBounds(23, 53, 91, 16);
197     lblUsernameValidation.setFont(new java.awt.Font("Dialog",0,10));
198 }
199 {
200     lblPermissionsValidation = new JLabel();
201     getContentPane().add(lblPermissionsValidation);
202     lblPermissionsValidation.setText("(at least one must be selected)");
203     lblPermissionsValidation.setBounds(120, 144, 175, 16);
204     lblPermissionsValidation.setFont(new java.awt.Font("Dialog",0,10));
205 }
206 {
207     lblPasswordValidation = new JLabel();
208     getContentPane().add(lblPasswordValidation);
209     lblPasswordValidation.setText("Max. 30 characters");
210     lblPasswordValidation.setBounds(25, 88, 95, 16);
211     lblPasswordValidation.setFont(new java.awt.Font("Dialog",0,10));
212 }
213 {
214     lblStatus = new JLabel();
215     getContentPane().add(lblStatus);
```

```
216         lblStatus.setText("Status:");
217         lblStatus.setBounds(74, 251, 40, 16);
218     }
219     {
220         chkEnabled = new JCheckBox();
221         getContentPane().add(chkEnabled);
222         chkEnabled.setText("Enabled");
223         chkEnabled.setBounds(128, 248, 70, 24);
224         chkEnabled.setSelected(user.isEnabled());
225     }
226     {
227         this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
228             ApplicationConstants.USERS_ICON)).getImage())
229         );
230         pack();
231         this.setSize(374, 364);
232     }
233
234 } catch (Exception e) {
235     e.printStackTrace();
236 }
237
238 }
239
240 /**
241 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
242 */
243
244 /**
245 * Checks if the information entered in the fields is correct.
246 * If so, the performUpdate() method is called.
247 * If the information is not correct, the admin user is alerted
248 */
249 public void editUser(){
250     if(txtUsername.getText()!= null && txtPassword.getText()!=null && txtFullName.getText()!= null
251         && (chkAdmin.isSelected() || chkEntry.isSelected() || chkExit.isSelected())){
```

```
252     try{
253         performUpdate();
254         JOptionPane.showMessageDialog(this, "The user has been updated",
255             "Information", JOptionPane.INFORMATION_MESSAGE);
256         this.dispose();
257     } catch(Exception e) {
258         JOptionPane.showMessageDialog(this, "The user has not been updated. \n" +
259             "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);
260         this.dispose();
261     }
262 } else {
263     JOptionPane.showMessageDialog(this, "Some fields are missing or the \n" +
264         "information entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);
265 }
266 }
267
268 /**
269 * Updates a user in the Users file by creating a temporary file and later renaming it
270 * @param user - the user to be updated
271 */
272 public void performUpdate() throws IOException {
273     try{
274
275         user.setPassword(txtPassword.getText());
276         user.setFullName(txtFullName.getText());
277         user.setAdmin(chkAdmin.isSelected());
278         user.setEntry(chkEntry.isSelected());
279         user.setExit(chkExit.isSelected());
280         user.setEnabled(chkEnabled.isSelected());
281
282         File file = new File(ApplicationConstants.USERS_FILE);
283         File tmpFile = new File(ApplicationConstants.USERS_TMP_FILE);
284         FileReader reader = new FileReader(file);
285         BufferedReader buff = new BufferedReader(reader);
286         FileWriter writer = new FileWriter(tmpFile);
287         BufferedWriter buffwriter = new BufferedWriter(writer);
```

```
288
289     boolean eof = false;           //stores if the end of the file (eof) has been reached
290
291     while(!eof){
292         String line = buff.readLine();
293         if(line == null){
294             eof = true;           //the end of the file has been reached
295         } else {
296             User userInFile = readUser(line);
297
298             if(userInFile.getUsername().equals(user.getUsername())){
299                 buffwriter.write(user.toString());
300                 buffwriter.newLine();
301             } else {
302                 buffwriter.write(userInFile.toString());
303                 buffwriter.newLine();
304             }
305
306         }
307     }
308
309     buffwriter.close();
310     buff.close();
311
312     file.delete();
313     tmpFile.renameTo(file);
314
315
316 } catch (Exception e){
317     System.out.println(e.toString());
318 }
319
320
321 }
322
323 /**

```

```
324     * Creates a User object from a tokenized String input
325     * @param line - a line read from the Users file
326     * @return - a User object
327     */
328     public User readUser(String line){
329         StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
330
331         if (tokenizer.countTokens() == 7){      //preliminary error checking
332             String username = tokenizer.nextToken();
333             String password = tokenizer.nextToken();
334             String fullName = tokenizer.nextToken();
335             boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
336             boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
337             boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
338             boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
339
340             return new User(username, password, fullName, admin, entry, exit, userEnabled);
341         }
342
343         return null;
344     }
345
346
347     /**
348     * Converts an int to a boolean
349     * @param 0 if false, 1 if true
350     * @return true if param is 1, else false
351     */
352     public boolean intToBoolean(int i){
353         return (i==1);
354     }
355
356 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the Group object that has information regarding item groups
5  * @author Alvaro Morales
6  * @date 24/06/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class Group {
13
14     /**
15      * The group code
16      */
17     private short code;
18
19     /**
20      * The group name
21      */
22     private String name;
23
24     /**
25      * Constructs a Group object
26      * @param code - the group code of type short
27      * @param name - the group name of type name
28      */
29     public Group(short code, String name){
30         this.code = code;
31         this.name = name;
32     }
33
34     /**
35      * Gets the group code
36      * @return the group code of type short
37     }
```

```
37     */
38     public short getCode() {
39         return code;
40     }
41
42     /**
43      * Sets the group code
44      * @param code, the group code of type short
45      */
46     public void setCode(short code) {
47         this.code = code;
48     }
49
50     /**
51      * Gets the group name
52      * @return the group name of type String
53      */
54     public String getName() {
55         return name;
56     }
57
58     /**
59      * Sets the group name
60      * @param name, the group name of type String
61      */
62     public void setName(String name) {
63         this.name = name;
64     }
65
66     /**
67      * Outputs the name of the group
68      */
69     public String toString(){
70         return getName();
71     }
72 }
```

```
1 import java.util.StringTokenizer;
2
3 /**
4 * -----
5 * Warehouse Application
6 * Constructs the Index object that stores information regarding the position of an Item in the Items file
7 * @author Alvaro Morales
8 * @date 24/06/2010
9 * @school Markham College
10 * @IDE Eclipse SDK
11 * @computer IBM ThinkPad R52
12 * -----
13 */
14 public class Index implements Comparable {
15
16     /**
17      * The position of the Item in the Items file (effectively the Item's ID)
18      */
19     private int position;
20
21     /**
22      * The field by which the Item is being indexed
23      * Index can be by Item Name or Code
24      */
25     private String field;
26
27     /**
28      * Constructs an Index object
29      * @param position - the position of the Item in the item's file (effectively the Item's ID)
30      */
31     public Index(int position, String field) {
32         this.position = position;
33         this.field = field;
34     }
35
36     /**
37      * Returns the position of the Item in the Items file
38      * @return the position of the Item in the Items file
39      */
40     public int getPosition() {
41         return position;
42     }
43
44     /**
45      * Sets the position of the Item in the Items file
46      * @param position the position of the Item in the Items file
47      */
48     public void setPosition(int position) {
49         this.position = position;
50     }
51
52     /**
53      * Returns the field by which the Item is being indexed
54      * @return the field by which the Item is being indexed
55      */
56     public String getField() {
57         return field;
58     }
59
60     /**
61      * Sets the field by which the Item is being indexed
62      * @param field the field by which the Item is being indexed
63      */
64     public void setField(String field) {
65         this.field = field;
66     }
67
68     /**
69      * Compares two Index objects based on their position
70      * @param other the other Index object
71      * @return -1 if this < other, 0 if this == other, 1 if this > other
72      */
73     public int compareTo(Index other) {
74         if (this.position < other.position) {
75             return -1;
76         } else if (this.position > other.position) {
77             return 1;
78         } else {
79             return 0;
80         }
81     }
82 }
```

```
37     * Constructs an Index object from a tokenized String from the index file.  
38     * Format: field|position  
39     * @param line - a tokenized String from the index file  
40     */  
41     public Index(String line){  
42         StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string  
43  
44         String field = tokenizer.nextToken();  
45         int position = new Integer(tokenizer.nextToken()).intValue();  
46  
47         this.position = position;  
48         this.field = field;  
49     }  
50  
51     /**  
52      * Gets the position of the item in the Items file  
53      * @return the position of the item in the Items file  
54      */  
55     public int getPosition() {  
56         return position;  
57     }  
58  
59     /**  
60      * Sets the position of the item in the Items file  
61      * @param position - the position in the items file, of type int  
62      */  
63     public void setPosition(int position) {  
64         this.position = position;  
65     }  
66  
67     /**  
68      * Gets the field by which the item is being indexed  
69      * @return the field by which the item is being indexed (code or name)  
70      */  
71     public String getField() {  
72         return field;
```

```
73     }
74
75     /**
76      * Sets the field by which the item is being indexed
77      * @param field - the field by which the item is being indexed, of type String
78      */
79     public void setField(String field) {
80         this.field = field;
81     }
82
83     /**
84      * Outputs the Index object formatted for the index file:
85      * Field + | + position.
86      * @return a String representation of the Index, formatted for the index file
87      */
88     public String toString(){
89         return new String(getField() + "|" + getPosition());
90     }
91
92     /**
93      * Compares an Index by comparings its field value using the method compareTo(String s)
94      * @param index - the index to compare
95      * @return 0 if they are equal, 1 if index is greater, -1 if index is smaller
96      */
97     public int compareTo(Index index){
98         return this.getField().compareTo(index.getField());
99     }
100
101    /**
102     * Compares an Index by comparings its field value using the method compareTo(String s)
103     * @param index - the index to compare
104     * @return 0 if they are equal,
105     * a value greater than 0 if index is greater,
106     * a value less than 0 if index is smaller
107     * @throws ClassCastException if the object being compared is not an instance of Index
108     */
```

```
109 public int compareTo(Object o) throws ClassCastException {
110     if (!(o instanceof Index)) {
111         throw new ClassCastException();
112     }
113
114     if (o instanceof Index) {
115         Index i = (Index) o;
116         int comparison = this.field.compareTo(i.getField());
117         return comparison;
118     }
119
120     return Integer.MIN_VALUE;      //if it is not an instance of Index
121 }
122
123 }
```

```
1 import java.util.*;
2
3 /**
4 * -----
5 * Warehouse Application
6 * Constructs the IndexBTree object, a binary tree of Item indexes
7 * @author Alvaro Morales
8 * @date 28/06/2010
9 * @school Markham College
10 * @IDE Eclipse SDK
11 * @computer IBM ThinkPad R52
12 * @mastery achieves HL mastery factors 12-15 with the implementation of a Binary Tree
13 * with all relevant key methods and full error checking
14 * -----
15 */
16 public class IndexBTree {
17
18     /**
19      * The root of the tree
20      */
21     private IndexTNode root;
22
23     /**
24      * Constructs an empty binary tree
25      */
26     public IndexBTree(){
27         this.root = null;
28     }
29
30     /**
31      * Constructs a binary tree
32      * @param root - the root of the binary tree
33      */
34     public IndexBTree(IndexTNode root) {
35         this.root = root;
36     }
```

```
37
38     /**
39      * Gets the root of the binary tree
40      * @return the root of the binary tree
41      */
42     public IndexTNode getRoot() {
43         return root;
44     }
45
46     /**
47      * Sets the root of the binary tree
48      * @param root - the root of the binary tree, of type IndexTNode
49      */
50     public void setRoot(IndexTNode root) {
51         this.root = root;
52     }
53
54     /**
55      * Checks if the tree is empty
56      * @return true if empty, else false
57      */
58     public boolean isEmpty(){
59         return (root == null);
60     }
61
62     /**
63      * Calculates the size of the binary tree, by recursively calling the method size(IndexTNode current)
64      * @return the size of the binary tree
65      */
66     public int size(){
67         return size(root);
68     }
69
70     /**
71      * Method is called recursively to calculate the number of nodes in a binary tree
72      * @param current - a pointer to the current node
```

```
73     * @return the size of the binary tree
74     */
75    public int size(IndexTNode current){
76        if(current == null){
77            return 0;
78        } else {
79            return 1 + size(current.getLeft()) + size(current.getRight());
80        }
81    }
82
83 /**
84 * Inserts an Index to the binary tree
85 * by recursively calling the method insertIndex(IndexTNode current, IndexTNode index)
86 * @param index - the Index to insert
87 */
88 public void insertIndex(IndexTNode index){
89     if(isEmpty()){
90         //Add dummy root to avoid the root (and consequently the entire tree) being deleted
91         String field = index.getIndex().getField().substring(0, 1);
92         IndexTNode dummyRoot = new IndexTNode(new Index(-1, field));
93         root = dummyRoot;
94         insertIndex(index);
95     } else {
96         insertIndex(root, index);
97     }
98 }
99
100 /**
101 * Method is called recursively to insert an index to the Index binary tree
102 * @param current - a pointer of the current node being evaluated for insertion
103 * @param index - the node to insert
104 * @mastery achieves HL mastery factor 4 by recursively calling itself to insert a node to the binary tree
105 */
106 private void insertIndex(IndexTNode current, IndexTNode index){
107     if(index.getIndex().getField().equals(current.getIndex().getField()) &&
108         index.getIndex().getPosition() == (current.getIndex().getPosition())){
```

```
109         //Error checking: if the Index already exists, do nothing
110         return;
111     } else if(index.getIndex().getField().compareTo(current.getIndex().getField()) < 0) {
112         if(current.getLeft() == null){
113             current.setLeft(index);
114         } else {
115             insertIndex(current.getLeft(), index);
116         }
117     } else if(index.getIndex().getField().compareTo(current.getIndex().getField()) > 0){
118         if(current.getRight() == null){
119             current.setRight(index);
120         } else {
121             insertIndex(current.getRight(), index);
122         }
123     }
124 }
125
126 /**
127 * Method gets a node from the Binary Tree given its field,
128 * by recursively calling the method getNode(IndexTNode current, String field)
129 * @param field - a node's field (item code or name)
130 * @return a node containing the field, or null if not found
131 */
132 public IndexTNode getNode(String field){
133     return getNode(root, field);
134 }
135
136 /**
137 * Method is called recursively to get a node from the binary tree given its field
138 * @param current - a pointer to the current node
139 * @param field - a node's field (item code or name)
140 * @return a node containing the field, or null if not found
141 */
142 private IndexTNode getNode(IndexTNode current, String field){
143     if (current == null){
144         return null;
```

```
145     } else {
146         if (current.getIndex().getField().compareTo(field) > 0){
147             return getNode(current.getLeft(), field);
148         } else if (current.getIndex().getField().compareTo(field) < 0){
149             return getNode(current.getRight(), field);
150         } else {
151             return current;
152         }
153     }
154 }
155
156 /**
157 * Searches the binary tree for a node exactly matching a query
158 * @param query - the exact match from an item name or code
159 * @return a node exactly matching a query; null if no results
160 */
161 public Item search(String query){
162     return search(root, query);
163 }
164
165 /**
166 * Traverses the binary tree to find a node that exactly matches a query
167 * @param current - a pointer to the current node
168 * @param query - the exact match from an item name or code
169 * @return a node exactly matching a query; null if no results
170 */
171 private Item search(IndexTNode current, String query){
172     if (current == null){
173         return null;
174     } else {
175         if (current.getIndex().getField().compareToIgnoreCase(query) > 0){
176             return search(current.getLeft(), query);
177         } else if (current.getIndex().getField().compareToIgnoreCase(query) < 0){
178             return search(current.getRight(), query);
179         } else if (current.getIndex().getField().equalsIgnoreCase(query) {
180             return Items.getItemFromFile(current.getIndex().getPosition());
```

```
181         }
182     }
183
184     return null;
185
186 }
187
188 /**
189 * Searches the binary tree for nodes partially matching a query
190 * @param query - the starting characters from an item name or code
191 * @return an ArrayList of search results, null if no results found
192 */
193 public ArrayList partialSearch(String query){
194
195     IndexBTree partialTree = getPartialTree(query);
196
197     if(partialTree == null) {
198         return null;
199     } else {
200         return getResults(partialTree, query);
201     }
202 }
203
204 /**
205 * Gets a sub-tree from the first node that matches the search criteria
206 * by calling the method getMatchingSubTree(IndexTNode current, String field)
207 * @param query - the starting characters from an item name or code
208 * @return an IndexBTree that has as its root the first node that matches the query,
209 * or null if no results
210 */
211 private IndexBTree getPartialTree(String query){
212     return getMatchingSubTree(root, query);
213 }
214
215 /**
216 * Gets a sub-tree from the first node that matches the search criteria
```

```
217     * @param current - a pointer to the current node
218     * @param query - the starting characters from an item name or code
219     * @return an IndexBTree that has as its root the first node that matches the query,
220     * or null if no results
221     */
222     private IndexBTree getMatchingSubTree(IndexTNode current, String query){
223         if(current == null){
224             return null;
225         }
226         if(query.length() < current.getIndex().getField().length()){
227             if (current.getIndex().getField().compareToIgnoreCase(query) == 0){
228                 return new IndexBTree(current);
229             } else if (current.getIndex().getField().toLowerCase().startsWith(query.toLowerCase())) {
230                 return new IndexBTree(current);
231             } else if (current.getIndex().getField().substring(
232                 0, query.length()).compareToIgnoreCase(query) < 0){
233                 return getMatchingSubTree(current.getRight(), query);
234             } else if (current.getIndex().getField().substring(
235                 0, query.length()).compareToIgnoreCase(query) > 0){
236                 return getMatchingSubTree(current.getLeft(), query);
237             } else {
238                 return null;
239             }
240         } else {
241             if(current == null){
242                 return null;
243             } else if (current.getIndex().getField().compareToIgnoreCase(query) == 0){
244                 return new IndexBTree(current);
245             } else if (current.getIndex().getField().toLowerCase().startsWith(query.toLowerCase())) {
246                 return new IndexBTree(current);
247             } else if (current.getIndex().getField().compareToIgnoreCase(
248                 query.substring(0, current.getIndex().getField().length())) == 0) {
249                 return new IndexBTree(current);
250             } else if (current.getIndex().getField().compareToIgnoreCase(
251                 query.substring(0, current.getIndex().getField().length())) < 0) {
252                 return getMatchingSubTree(current.getRight(), query);
```

```
253     } else if (current.getIndex().getField().compareToIgnoreCase(
254         query.substring(0, current.getIndex().getField().length())) > 0) {
255         return getMatchingSubTree(current.getLeft(), query);
256     } else {
257         return null;
258     }
259 }
260 }
261 /**
262 * Returns an ArrayList of Items that match the query
263 * @param tree - a sub-tree matching that has the first node that matched the query as its root
264 * @param query - the starting characters from an item name or code
265 * @return an ArrayList of Items that match the query
266 */
267 private ArrayList<Item> getResults(IndexBTree tree, String query){
268     ArrayList<Item> searchResults = new ArrayList();
269     addResults(tree.getRoot(), query, searchResults);
270
271     if(!searchResults.isEmpty()){
272         searchResults.trimToSize();
273         return searchResults;
274     } else {
275         return null;
276     }
277 }
278 }
279 /**
280 * Traverses the tree and adds nodes that match the query to the ArrayList of search results
281 * @param current - a pointer to the current node
282 * @param query - the starting characters from an item name or code
283 * @param searchResults - an ArrayList of Items matching the query
284 */
285 private void addResults(IndexTNode current, String query, ArrayList<Item> searchResults){
286     if(current == null){
```

```
289         return;
290     } else if(current.getIndex().getPosition() == -1){
291         //If dummy root, recursively call methods on children
292         addResults(current.getLeft(), query, searchResults);
293         addResults(current.getRight(), query, searchResults);
294     } else if (!current.getIndex().getField().toLowerCase().startsWith(query.toLowerCase())){
295         if(current.getLeft() != null &&
296             current.getLeft().getIndex().getField().toLowerCase().startsWith(query.toLowerCase())){
297             addResults(current.getLeft(), query, searchResults);
298         } else if(current.getRight() != null &&
299             current.getRight().getIndex().getField().toLowerCase().startsWith(
300                 query.toLowerCase())){
301             addResults(current.getRight(), query, searchResults);
302         } else {
303             return;
304         }
305     } else {
306         addResults(current.getLeft(), query, searchResults);
307         searchResults.add(Items.getItemFromFile(current.getIndex().getPosition()));
308         addResults(current.getRight(), query, searchResults);
309     }
310 }
311
312 /**
313 * Gets the parent of a node
314 * by recursively calling the method getParent(IndexTNode current, IndexTNode node, String field)
315 * @param node - the child node
316 * @return the parent of the node
317 */
318 public IndexTNode getParent(IndexTNode node){
319     return getParent(root, node, node.getIndex().getField());
320 }
321
322
323 /**
324 * Method is called recursively to find a node's parent
```

```
325     * @param current - a pointer to the current node being evaluated
326     * @param node - the child node
327     * @param field - the data that the node stores
328     * @return the node's parent
329     */
330    private IndexTNode getParent(IndexTNode current, IndexTNode node, String field){
331
332        if(current == null){
333            return null;
334        } else if (current.getIndex().getField().equals(field) && current.equals(getRoot())) {
335            return null;
336        } else if (current.getLeft() != null && current.getLeft().equals(node)){
337            return current;
338        } else if (current.getLeft() != null && current.getRight().equals(node)){
339            return current;
340        } else if (current.getIndex().getField().compareTo(field) < 0){
341            return getParent(current.getRight(), node, field);
342        } else if (current.getIndex().getField().compareTo(field) > 0){
343            return getParent(current.getLeft(), node, field);
344        } else {
345            return null;
346        }
347
348    }
349
350    /**
351     * Deletes an Index node from the binary tree
352     * @param node - the node to delete
353     */
354    public void deleteIndex(IndexTNode node){
355        if (node.equals(getRoot()) && node.getLeft() == null && node.getRight() == null){
356            setRoot(null);
357        } else {
358            if(node.getIndex().getField().compareTo(getParent(node).getIndex().getField()) > 0){
359                //If it is the right child of its parent
360                if(node.getRight() != null){
```

```
361         getParent(node).setRight(node.getRight());
362     }
363     if(node.getLeft() != null){
364         node.getRight().setLeft(node.getLeft());
365     }
366     if(node.getLeft() == null && node.getRight() == null){
367         getParent(node).setRight(null);
368     }
369     node.setLeft(null);
370     node.setRight(null);
371 } else { //If it is the left child of its parent
372     if (node.getLeft() != null){
373         getParent(node).setLeft(node.getLeft());
374     }
375     if (node.getRight() != null){
376         node.getLeft().setRight(node.getRight());
377     }
378     if(node.getLeft() == null && node.getRight() == null){
379         getParent(node).setLeft(null);
380     }
381     node.setLeft(null);
382     node.setRight(null);
383 }
384 }
385 }
386
387
388 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the IndexTreeNode object, a node in the Index Binary Tree
5  * @author Alvaro Morales
6  * @date 28/06/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class IndexTNode {
13
14     /**
15      * The Index object, an index of an Item
16      */
17     private Index index;
18
19     /**
20      * This node's left child
21      */
22     private IndexTNode left;
23
24     /**
25      * This node's right child
26      */
27     private IndexTNode right;
28
29     /**
30      * Constructs an IndexTreeNode object
31      * @param index - the index of an Item
32      */
33     public IndexTNode(Index index) {
34         this.index = index;
35     }
36 }
```

```
37     /**
38      * Gets the Item index
39      * @return the item index
40      */
41     public Index getIndex() {
42         return index;
43     }
44
45     /**
46      * Sets the Item index
47      * @param index - the index of an item
48      */
49     public void setIndex(Index index) {
50         this.index = index;
51     }
52
53     /**
54      * Gets the node's left child
55      * @return the node's left child
56      */
57     public IndexTNode getLeft() {
58         return left;
59     }
60
61     /**
62      * Sets the node's left child
63      * @param left - the node's left child, of type IndexTreeNode
64      */
65     public void setLeft(IndexTNode left) {
66         this.left = left;
67     }
68
69     /**
70      * Gets the node's right child
71      * @return the node's right child
72      */
```

```
73     public IndexTNode getRight() {
74         return right;
75     }
76
77     /**
78      * Sets the node's right child
79      * @param right - the node's right child, of type IndexTreeNode
80      */
81     public void setRight(IndexTNode right) {
82         this.right = right;
83     }
84
85 }
```

```
1 import java.io.*;
2 import java.util
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the Item object that has information regarding an item stored inside the warehouse
8 * @author Alvaro Morales
9 * @date 24/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * @mastery achieves HL mastery factor 8 by having private instance variables
14 * and providing appropriate getters and setters (encapsulation)
15 * @mastery achieves SL mastery factor 2 as it is a user defined method with attributes and operations
16 * @mastery achieves SL mastery factor 3 as the Item object is used as a data record to store an item
17 * -----
18 */
19
20 public class Item {
21
22     /**
23      * The ID of the item in the internal system
24      */
25     private int ID;
26
27     /**
28      * The code of the item in the existing system
29      * Stored as a String to allow leading 0's
30      */
31     private String code;
32
33     /**
34      * The item's name
35      */
36     private String name;
```

```
37
38     /**
39      * The Group an item belongs to
40      */
41     private Group group;
42
43     /**
44      * The Unit of Measurement (UM) by which the item is measured and stored
45      */
46     private UM um;
47
48     /**
49      * The item's description
50      */
51     private String description;
52
53     /**
54      * The Location of the item inside the warehouse
55      */
56     private Location location;
57
58     /**
59      * The username of the user that created the Item
60      */
61     private String createdBy;
62
63     /**
64      * The Item's creation date
65      */
66     private GregorianCalendar creationDate;
67
68     /**
69      * A comparator based on location
70      */
71     public static final Comparator LOCATION_ORDER = new Comparator() {
72         public int compare(Object obj1, Object obj2){
```

```
73         return ((Item)obj1).getLocation().compareTo(((Item)obj2).getLocation());
74     }
75 };
76 /**
77 * Constructs an Item object
78 * @param ID - the ID of the item in the internal system
79 * @param code - the code of the item in the existing system
80 * @param name - the item's name
81 * @param group - the Group an item belongs to
82 * @param um - the Unit of Measurement (UM) by which the item is measured and stored
83 * @param description - the item's description
84 * @param location - the Location of the item inside the warehouse
85 */
86 public Item(int ID, String code, String name, Group group, UM um, String description,
87             Location location, String createdBy, GregorianCalendar creationDate){
88     this.ID = ID;
89     this.code = code;
90     this.name = name;
91     this.group = group;
92     this.um = um;
93     this.description = description;
94     this.location = location;
95     this.createdBy = createdBy;
96     this.creationDate = creationDate;
97 }
98 /**
99 * Gets the Item's code
100 * @return the item code
101 */
102 public String getCode() {
103     return code;
104 }
105 }
106 /**
107 */
108 */
```

```
109     * Sets the item's code
110     * @param code - the item code (in the existing system), of type String
111     */
112     public void setCode(String code) {
113         this.code = code;
114     }
115
116     /**
117      * Gets the item description
118      * @return the item description
119      */
120     public String getDescription() {
121         return description;
122     }
123
124     /**
125      * Sets the item description
126      * @param description - the item description, of type String
127      */
128     public void setDescription(String description) {
129         this.description = description;
130     }
131
132     /**
133      * Gets the group the item belongs to
134      * @return the group the item belongs to
135      */
136     public Group getGroup() {
137         return group;
138     }
139
140     /**
141      * Gets the name of the user that created this item
142      * @return the username of the user that created this item
143      */
144     public String getCreatedBy() {
```

```
145     return createdBy;
146 }
147 /**
148 * Sets the name of the user that created this item
149 * @param createdBy - the username of the user that created this item, of type String
150 */
151 public void setCreatedBy(String createdBy) {
152     this.createdBy = createdBy;
153 }
154
155 /**
156 * Gets the item's creation date
157 * @return the item's creation date
158 */
159 public GregorianCalendar getCreationDate() {
160     return creationDate;
161 }
162
163 /**
164 * Sets the item's creation date
165 * @param creationDate - the item's creation date, of type GregorianCalendar
166 */
167 public void setCreationDate(GregorianCalendar creationDate) {
168     this.creationDate = creationDate;
169 }
170
171 /**
172 * Sets the group an item belongs to
173 * @param group - the group an item belongs to, of type Group
174 */
175 public void setGroup(Group group) {
176     this.group = group;
177 }
178
179 /**
180 *
```

```
181     * Gets the item ID (of the internal system)
182     * @return the item ID
183     */
184     public int getID() {
185         return ID;
186     }
187
188     /**
189      * Sets the item ID
190      * @param id - the item ID, of type int
191      */
192     public void setID(int id) {
193         this.ID = id;
194     }
195
196     /**
197      * Gets the Location of the item stored in the warehouse
198      * @return the item location
199      */
200     public Location getLocation() {
201         return location;
202     }
203
204     /**
205      * Sets the Location of where the item is stored in the warehouse
206      * @param location
207      */
208     public void setLocation(Location location) {
209         this.location = location;
210     }
211
212     /**
213      * Gets the item name
214      * @return the item name
215      */
216     public String getName() {
```

```
217     return name;
218 }
219
220 /**
221 * Sets the item name
222 * @param name - the item name, of type String
223 */
224 public void setName(String name) {
225     this.name = name;
226 }
227
228 /**
229 * Gets the Unit of Measurement (UM) by which the item is stored/measured
230 * @return the unit of measurement
231 */
232 public UM getUm() {
233     return um;
234 }
235
236 /**
237 * Sets the Unit of Measurement (UM) by which the item is stored/measured
238 * @param um - the unit of measurement, of type UM
239 */
240 public void setUm(UM um) {
241     this.um = um;
242 }
243
244 /**
245 * Writes the Item object to the Items Random Access File
246 * @param file - the Items Random Access File
247 */
248 public void writeItemToFile(RandomAccessFile file){
249     writeIDToFile(file);
250     writeCodeToFile(file);
251     writeNameToFile(file);
252     writeGroupToFile(file);
```

```
253     writeUMToFile(file);
254     writeDescriptionToFile(file);
255     writeLocationToFile(file);
256     writeCreatedBy(file);
257     writeCreationDate(file);
258 }
259
260 /**
261 * Writes the Item ID to the Items Random Access File
262 * @param file - the Items Random Access File
263 */
264 private void writeIDToFile(RandomAccessFile file){
265     try{
266         file.getFilePointer();
267         file.writeInt(getID());
268     } catch (Exception e) {
269     }
270 }
271
272 /**
273 * Writes the Item Code to the Items Random Access File
274 * @param file - the Items Random Access File
275 */
276 private void writeCodeToFile(RandomAccessFile file){
277     String code = getCode();
278     while(code.length()!= 6){
279         code = "0" + code;
280     }
281
282     try{
283         file.writeUTF(code);
284     } catch (Exception e) {
285     }
286 }
287
288 }
```

```
289     }
290
291     /**
292      * Writes the Item Name to the Items Random Access File
293      * @param file - the Items Random Access File
294      */
295     private void writeNameToFile(RandomAccessFile file){
296         String name = getName();
297
298         while(name.length()!= 200){
299             name = name + " ";
300         }
301
302         try{
303             file.writeUTF(name);
304         } catch (Exception e) {
305
306         }
307     }
308
309     /**
310      * Writes the Item Group to the Items Random Access File
311      * @param file - the Items Random Access File
312      */
313     private void writeGroupToFile(RandomAccessFile file){
314         try{
315             file.writeShort(getGroup().getCode());
316         } catch (Exception e) {
317
318         }
319     }
320
321     /**
322      * Writes the Item UM to the Items Random Access File
323      * @param file - the Items Random Access File
324      */
```

```
325     private void writeUMToFile(RandomAccessFile file){  
326         try{  
327             file.writeShort(getUm().getCode());  
328         } catch (Exception e) {  
329             }  
330         }  
331     }  
332  
333     /**  
334      * Writes the Item description to the Items Random Access File  
335      * @param file - the Items Random Access File  
336      */  
337     private void writeDescriptionToFile(RandomAccessFile file){  
338         String description = getDescription();  
339  
340         if(description == null || description.equals("")){           //As description is optional  
341             description = "<No description>";  
342         }  
343  
344         while(description.length()!= 200){  
345             description = description + " ";  
346         }  
347  
348         try{  
349             file.writeUTF(description);  
350         } catch (Exception e) {  
351             }  
352         }  
353     }  
354  
355     /**  
356      * Writes the Item location to the Items Random Access File  
357      * @param file - the Item's Random Access File  
358      */  
359     private void writeLocationToFile(RandomAccessFile file){  
360 }
```

```
361     try{
362         file.writeByte(1);                                //Write the warehouse number
363         file.writeByte(getLocation().getAisle());        //Write the aisle number
364         file.writeByte(getLocation().getColumn());       //Write the column number
365         file.writeChar(getLocation().getRow());          //Write the row number
366     } catch (Exception e) {
367
368     }
369 }
370
371 /**
372 * Writes the username of the user that created the Item to the Items Random Access File
373 * @param file- the Item's Random Access File
374 */
375 private void writeCreatedBy(RandomAccessFile file){
376     String createdBy = getCreatedBy();
377
378     try{
379         while(createdBy.length()!= 20){
380             createdBy = createdBy + " ";
381         }
382
383         file.writeUTF(createdBy);
384     } catch (Exception e){
385
386     }
387 }
388
389 /**
390 * Writes the Item's creation date to the Items Random Access File
391 * @param file - the Item's Random Access File
392 */
393 private void writeCreationDate(RandomAccessFile file){
394     try{
395         file.writeShort(getCreationDate().get(Calendar.YEAR));
396         file.writeByte(getCreationDate().get(Calendar.MONTH));
```

```
397         file.writeByte(getCreationDate().get(Calendar.DAY_OF_MONTH));
398     } catch (Exception e){
399
400     }
401 }
402 /**
403 * Outputs the item in the format Code: Name
404 */
405 public String toString(){
406     return getCode() + ":" + getName();
407 }
408
409
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.io.*;
4 import java.util.*;
5 import javax.swing.border.*;
6 import javax.swing.event.*;
7 import javax.swing.table.*;
8
9 /**
10 * -----
11 * Warehouse Application
12 * The Item Description screen to display, edit or delete the item and view transactions
13 * @author Alvaro Morales
14 * @date 04/07/2010
15 * @school Markham College
16 * @IDE Eclipse SDK
17 * @computer IBM ThinkPad R52
18 * -----
19 */
20 public class ItemDescription extends javax.swing.JFrame {
21
22     /**
23      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
24      */
25
26     {
27         //Set Look & Feel
28         try {
29             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     /**
36      * GUI components generated by Jigloo
```

```
37     */
38
39     private JTabbedPane tabContainer;
40     private JPanel tabDescription;
41     private JPanel tabRecords;
42     private JLabel lblItemInformation;
43     private JTextField txtLocation;
44     private JTextField txtDate;
45     private JTextField txtCreatedBy;
46     private JTextArea txtDescription;
47     private JComboBox cmbGroup;
48     private JTextArea txtName;
49     private JLabel lblColumn;
50     private JComboBox cmbColumn;
51     private JLabel lblRow;
52     private JComboBox cmbRow;
53     private JComboBox cmbAisle;
54     private JLabel lblAisle;
55     private JLabel lblDescriptionInfo;
56     private JLabel lblOptional;
57     private JLabel lblNameInfo;
58     private JButton btnSearch;
59     private JButton btnDeleteTransaction;
60     private JTextField txtUM;
61     private JTextField txtGroup;
62     private JLabel lblUM;
63     private JComboBox cmbUM;
64     private JLabel lblCodeInfo;
65     private JLabel lblCode;
66     private JTextField txtCode;
67     private JLabel lblRequired;
68     private JSeparator spRecords;
69     private JComboBox cmbYearUntil;
70     private JComboBox cmbMonthUntil;
71     private JComboBox cmbDayUntil;
72     private JLabel lblLookUntil;
```

```
73     private JComboBox cmbYear;
74     private JComboBox cmbMonth;
75     private JComboBox cmbDay;
76     private JLabel lblLookFrom;
77     private JLabel lblInfo2;
78     private JLabel lblInfo1;
79     private JLabel lblInfo;
80     private JTextField txtTotal;
81     private JLabel lblTotal;
82     private JTextField txtUmStockInfo;
83     private JLabel lblUnitOfMeasurement;
84     private JTable tblTransactions;
85     private JScrollPane scpRecords;
86     private JLabel lblItemRecords;
87     private JSeparator spImage;
88     private JButton btnChange;
89     private JLabel lblItemImage;
90     private JLabel lblImage;
91     private JPanel tabImage;
92     private JSeparator spOptions;
93     private JButton btnDelete;
94     private JButton btnEditing;
95     private JLabel lblCreationDetails;
96     private JSeparator spCreation;
97     private JLabel lblLocation;
98     private JLabel lblDate;
99     private JLabel lblCreatedBy;
100    private JLabel lblDescription;
101    private JLabel lblGroup;
102    private JLabel lblName;
103
104    /**
105     * The item that is being displayed.
106     * The information in this screen belongs to this item.
107     */
108    private Item item;
```

```
109
110     /**
111      * The user that logged in
112      */
113     private User user;
114
115     /**
116      * A linked list of this item's transactions
117      */
118     private TransactionList list;
119
120     /**
121      * The current date
122      */
123     private GregorianCalendar currentDate;
124
125     /**
126      * The lower boundary for the date interval of transactions to display
127      */
128     private GregorianCalendar dateStart;
129
130     /**
131      * The upper boundary for the date interval of transactions to display
132      */
133     private GregorianCalendar dateEnd;
134
135     /**
136      * Constructs the ItemDescription screen
137      * @param item - the item that is being displayed
138      * @param user - the user that logged in
139      */
140     public ItemDescription(Item item, User user) {
141         super();
142         this.item = item;
143         this.user = user;
144         currentDate = new GregorianCalendar();
```

```
145     dateStart = new GregorianCalendar(ApplicationConstants.TRANSACTION_YEAR_START,1,1);      //Start date
146     dateEnd = currentDate;
147     populateList();
148     initGUI();
149     loadFields();
150 }
151
152 /**
153 * Constructs the ItemDescriptionScreen that only DISPLAYS the item image.
154 * For use in transactions processing
155 */
156 public ItemDescription(Item item){
157     super();
158     this.item = item;
159     initGUIONlyImage();
160 }
161
162 /**
163 * Jigloo-generated method that initializes the GUI and components
164 */
165 private void initGUI() {
166     try {
167         this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
168
169         String title = item.getName();
170
171         if(title.length() >= 40){
172             title = title.substring(0, 40) + "...";
173         }
174
175         this.setTitle(title);
176         this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
177             ApplicationConstants.WAREHOUSE_LOGO)).getImage());
178     };
179     getContentPane().setLayout(null);
180     this.setResizable(false);
```

```
181 {
182     tabContainer = new JTabbedPane();
183     getContentPane().add(tabContainer, "Center");
184     tabContainer.setBounds(0, 0, 522, 527);
185     {
186         tabDescription = new JPanel();
187         tabContainer.addTab("Description", tabDescription);
188         tabDescription.setLayout(null);
189         tabDescription.setPreferredSize(new java.awt.Dimension(517, 547));
190         tabDescription.setSize(482, 541);
191         {
192             lblName = new JLabel();
193             tabDescription.add(lblName);
194             lblName.setText("Name:");
195             lblName.setBounds(71, 75, 36, 13);
196         }
197         {
198             lblGroup = new JLabel();
199             tabDescription.add(lblGroup);
200             lblGroup.setText("Group:");
201             lblGroup.setBounds(16, 178, 41, 13);
202         }
203         {
204             lblDescription = new JLabel();
205             tabDescription.add(lblDescription);
206             lblDescription.setText("Description:");
207             lblDescription.setBounds(39, 210, 69, 17);
208         }
209         {
210             lblCreatedBy = new JLabel();
211             tabDescription.add(lblCreatedBy);
212             lblCreatedBy.setText("Created By:");
213             lblCreatedBy.setBounds(42, 411, 65, 13);
214         }
215         {
216             lblDate = new JLabel();
```

```
217     tabDescription.add(lblDate);
218     lblDate.setText("On:");
219     lblDate.setBounds(287, 410, 30, 15);
220 }
221 {
222     lblLocation = new JLabel();
223     tabDescription.add(lblLocation);
224     lblLocation.setText("Location:");
225     lblLocation.setBounds(55, 324, 52, 14);
226 }
227 {
228     txtName = new JTextArea();
229     tabDescription.add(txtName);
230     txtName.setBounds(132, 77, 353, 82);
231     txtName.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
232     txtName.setEditable(false);
233     txtName.setLineWrap(true);
234     txtName.setDocument(new TextFieldLimit(200));
235 }
236 {
237     ComboBoxModel cmbFamilyModel = new DefaultComboBoxModel(MainScreen.groups);
238     cmbGroup = new JComboBox();
239     tabDescription.add(cmbGroup);
240     cmbGroup.setModel(cmbFamilyModel);
241     cmbGroup.setBounds(64, 172, 186, 25);
242     cmbGroup.setVisible(false);
243 }
244 {
245     txtDescription = new JTextArea();
246     tabDescription.add(txtDescription);
247     txtDescription.setBounds(132, 210, 359, 86);
248     txtDescription.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
249     txtDescription.setEditable(false);
250     txtDescription.setLineWrap(true);
251     txtDescription.setDocument(new TextFieldLimit(200));
252 }
```

```
253 {
254     txtCreatedBy = new JTextField();
255     tabDescription.add(txtCreatedBy);
256     txtCreatedBy.setBounds(119, 409, 150, 20);
257     txtCreatedBy.setEditable(false);
258 }
259 {
260     txtDate = new JTextField();
261     tabDescription.add(txtDate);
262     txtDate.setBounds(328, 409, 146, 18);
263     txtDate.setEditable(false);
264 }
265 {
266     txtLocation = new JTextField();
267     tabDescription.add(txtLocation);
268     txtLocation.setBounds(132, 321, 192, 19);
269     txtLocation.setEditable(false);
270     txtLocation.setSize(192, 20);
271     txtLocation.setEnabled(true);
272     txtLocation.setEditable(false);
273     txtLocation.setVisible(true);
274 }
275 {
276     lblItemInformation = new JLabel();
277     tabDescription.add(lblItemInformation);
278     lblItemInformation.setText("Item Information");
279     lblItemInformation.setFont(new java.awt.Font("Dialog",1,14));
280     lblItemInformation.setBounds(12, 12, 120, 16);
281 }
282 {
283     spCreation = new JSeparator();
284     tabDescription.add(spCreation);
285     spCreation.setBounds(12, 364, 493, 7);
286 }
287 {
288     lblCreationDetails = new JLabel();
```

```
289     tabDescription.add(lblCreationDetails);
290     lblCreationDetails.setText("Creation Details");
291     lblCreationDetails.setFont(new java.awt.Font("Dialog",1,14));
292     lblCreationDetails.setBounds(12, 378, 120, 13);
293 }
294
295 //Checks for user permissions:
296 //only users with Admin or Entry permissions can change/delete an item
297 if(user.isAdmin() || user.isEntry()){
298 {
299     btnEditing = new JButton();
300     tabDescription.add(btnEditing);
301     btnEditing.setText("Turn Editing On");
302     btnEditing.setBounds(131, 454, 120, 26);
303     btnEditing.addActionListener(new ActionListener() {
304         public void actionPerformed (ActionEvent e){
305             enableEditing();
306         }
307     });
308 }
309 }
310
311 {
312     btnDelete = new JButton();
313     tabDescription.add(btnDelete);
314     btnDelete.setText("Delete Item");
315     btnDelete.setBounds(287, 454, 98, 26);
316     btnDelete.addActionListener(new ActionListener() {
317         public void actionPerformed (ActionEvent e){
318             performDelete();
319         }
320     });
321 }
322 }
323 {
324     spOptions = new JSeparator();
```

```
325         tabDescription.add(spOptions);
326         spOptions.setBounds(12, 441, 493, 13);
327     }
328     {
329         lblRequired = new JLabel();
330         tabDescription.add(lblRequired);
331         lblRequired.setText("Required");
332         lblRequired.setBounds(64, 95, 54, 13);
333         lblRequired.setFont(new java.awt.Font("Dialog",0,10));
334         lblRequired.setVisible(false);
335     }
336     {
337         lblNameInfo = new JLabel();
338         tabDescription.add(lblNameInfo);
339         lblNameInfo.setText("Max. 200 characters");
340         lblNameInfo.setBounds(12, 107, 98, 15);
341         lblNameInfo.setFont(new java.awt.Font("Dialog",0,10));
342         lblNameInfo.setVisible(false);
343     }
344     {
345         lblOptional = new JLabel();
346         tabDescription.add(lblOptional);
347         lblOptional.setText("Optional");
348         lblOptional.setBounds(67, 233, 47, 12);
349         lblOptional.setFont(new java.awt.Font("Dialog",0,10));
350         lblOptional.setVisible(false);
351     }
352     {
353         lblDescriptionInfo = new JLabel();
354         tabDescription.add(lblDescriptionInfo);
355         lblDescriptionInfo.setText("Max. 200 characters");
356         lblDescriptionInfo.setBounds(12, 248, 96, 17);
357         lblDescriptionInfo.setFont(new java.awt.Font("Dialog",0,10));
358         lblDescriptionInfo.setVisible(false);
359     }
360     {
```

```
361     lblAisle = new JLabel();
362     tabDescription.add(lblAisle);
363     lblAisle.setText("Aisle");
364     lblAisle.setBounds(152, 302, 28, 16);
365     lblAisle.setVisible(false);
366 }
367 {
368     ComboBoxModel cmbAisleModel =
369         new DefaultComboBoxModel(MainScreen.locations.getAisles());
370     cmbAisle = new JComboBox();
371     tabDescription.add(cmbAisle);
372     cmbAisle.setModel(cmbAisleModel);
373     cmbAisle.setBounds(135, 324, 63, 25);
374     cmbAisle.setVisible(false);
375 }
376 {
377     ComboBoxModel cmbRowModel = new DefaultComboBoxModel(
378             MainScreen.locations.getRows());
379     cmbRow = new JComboBox();
380     tabDescription.add(cmbRow);
381     cmbRow.setModel(cmbRowModel);
382     cmbRow.setBounds(214, 324, 67, 25);
383     cmbRow.setVisible(false);
384 }
385 {
386     lblRow = new JLabel();
387     tabDescription.add(lblRow);
388     lblRow.setText("Row");
389     lblRow.setBounds(235, 301, 25, 17);
390     lblRow.setVisible(false);
391 }
392 {
393     ComboBoxModel cmbColumnModel = new DefaultComboBoxModel(
394             MainScreen.locations.getColumns());
395     cmbColumn = new JComboBox();
396     tabDescription.add(cmbColumn);
```

```
397     cmbColumn.setModel(cmbColumnModel);
398     cmbColumn.setBounds(299, 324, 67, 25);
399     cmbColumn.setVisible(false);
400 }
401 {
402     lblColumn = new JLabel();
403     tabDescription.add(lblColumn);
404     lblColumn.setText("Column");
405     lblColumn.setBounds(311, 302, 43, 16);
406     lblColumn.setVisible(false);
407 }
408 {
409     txtCode = new JTextField();
410     tabDescription.add(txtCode);
411     txtCode.setBounds(132, 40, 194, 24);
412     txtCode.setEditable(false);
413     txtCode.setDocument(new TextFieldDigitLimit(6));
414 }
415 {
416     lblCode = new JLabel();
417     tabDescription.add(lblCode);
418     lblCode.setText("Code:");
419     lblCode.setBounds(74, 44, 35, 16);
420 }
421 {
422     lblCodeInfo = new JLabel();
423     tabDescription.add(lblCodeInfo);
424     lblCodeInfo.setText("Required. Must be exactly 6 numbers");
425     lblCodeInfo.setBounds(332, 44, 178, 16);
426     lblCodeInfo.setFont(new java.awt.Font("Dialog",0,10));
427     lblCodeInfo.setVisible(false);
428 }
429 {
430     ComboBoxModel cmbUMModel = new DefaultComboBoxModel(
431         MainScreen.umArray);
432     cmbUM = new JComboBox();
```

```
433         tabDescription.add(cmbUM);
434         cmbUM.setModel(cmbUMModel);
435         cmbUM.setBounds(308, 172, 195, 25);
436         cmbUM.setVisible(false);
437     }
438     {
439         lblUM = new JLabel();
440         tabDescription.add(lblUM);
441         tabDescription.add(getTxtGroup());
442         tabDescription.add(getJTextField1());
443         lblUM.setText("U.M. :");
444         lblUM.setBounds(266, 176, 30, 16);
445     }
446 }
447 {
448     tabImage = new JPanel();
449     tabContainer.addTab("Image", tabImage);
450     tabImage.setLayout(null);
451     {
452         lblImage = new JLabel();
453         tabImage.add(lblImage);
454         lblImage.setBounds(103, 52, 320, 240);
455         lblImage.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
456         try {
457             lblImage.setIcon(
458                 new ImageIcon(getClass().getClassLoader().getResource(
459                     "Files/Item_Images/" + item.getID() + ".jpg")));
460         } catch (Exception e) {
461             //Item has no Image
462             lblImage.setIcon(
463                 new ImageIcon(getClass().getClassLoader().getResource(
464                     "Files/Item_Images/NoImage.jpg")));
465         }
466     }
467     {
468         lblItemImage = new JLabel();
```

```
469         tabImage.add(lblItemImage);
470         lblItemImage.setText("Item Image");
471         lblItemImage.setFont(new java.awt.Font("Dialog",1,14));
472         lblItemImage.setBounds(12, 18, 493, 16);
473         lblItemImage.setAlignmentX(0.5f);
474         lblItemImage.setHorizontalTextPosition(SwingConstants.CENTER);
475     }
476     {
477         btnChange = new JButton();
478         tabImage.add(btnChange);
479         btnChange.setText("Change Picture");
480         btnChange.setBounds(205, 304, 121, 26);
481         btnChange.addActionListener(new ActionListener() {
482             public void actionPerformed (ActionEvent e){
483                 changeItemImage();
484             }
485         });
486     }
487     {
488         spImage = new JSeparator();
489         tabImage.add(spImage);
490         spImage.setBounds(12, 344, 493, 10);
491     }
492 }
493 {
494     tabRecords = new JPanel();
495     tabContainer.addTab("Stock Information", null, tabRecords, null);
496     tabRecords.setLayout(null);
497     {
498         lblItemRecords = new JLabel();
499         tabRecords.add(lblItemRecords);
500         lblItemRecords.setText("Item Records");
501         lblItemRecords.setFont(new java.awt.Font("Dialog",1,14));
502         lblItemRecords.setBounds(12, 12, 101, 16);
503     }
504 }
```

```
505     scpRecords = new JScrollPane();
506     tabRecords.add(scpRecords);
507     scpRecords.setBounds(44, 177, 431, 194);
508     {
509         ArrayList transactionsToDisplay = list.searchTransactions(dateStart, dateEnd);
510         TransactionsTableModel tblTransactionsModel =
511             new TransactionsTableModel(transactionsToDisplay);
512         tblTransactions = new JTable();
513         scpRecords.setViewportView(tblTransactions);
514         tblTransactions.setModel(tblTransactionsModel);
515         tblTransactions.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
516         setColumnWidths();
517         tblTransactions.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
518         if(user.isAdmin()){
519             tblTransactions.getSelectionModel().addListSelectionListener(
520                 new ListSelectionListener(){
521                     public void valueChanged(ListSelectionEvent e){
522                         enableDeleteTransaction(e.getFirstIndex());
523                     }
524                 });
525             }
526         }
527     }
528     {
529         lblUnitOfMeasurement = new JLabel();
530         tabRecords.add(lblUnitOfMeasurement);
531         lblUnitOfMeasurement.setText("U.M. :");
532         lblUnitOfMeasurement.setBounds(144, 143, 30, 16);
533     }
534     {
535         txtUmStockInfo = new JTextField();
536         tabRecords.add(txtUmStockInfo);
537         txtUmStockInfo.setBounds(186, 141, 180, 20);
538         txtUmStockInfo.setEditable(false);
539         txtUmStockInfo.setText(item.getUm().toString());
540     }
```

```
541 {
542     lblTotal = new JLabel();
543     tabRecords.add(lblTotal);
544     lblTotal.setText("Current Balance:");
545     lblTotal.setBounds(44, 389, 101, 21);
546 }
547 {
548     txtTotal = new JTextField();
549     tabRecords.add(txtTotal);
550     txtTotal.setBounds(150, 389, 180, 20);
551     txtTotal.setEditable(false);
552     txtTotal.setText("") + Transactions.getQuantityInStock(item));
553 }
554 {
555     lblInfo = new JLabel();
556     tabRecords.add(lblInfo);
557     lblInfo.setText("Information");
558     lblInfo.setFont(new java.awt.Font("Dialog",1,12));
559     lblInfo.setBounds(44, 428, 65, 16);
560 }
561 {
562     lblInfo1 = new JLabel();
563     tabRecords.add(lblInfo1);
564     lblInfo1.setText("For entries, the 'Document' column "
565                     + "gives the number of the Proof of Reception");
566     lblInfo1.setFont(new java.awt.Font("Dialog",0,10));
567     lblInfo1.setBounds(44, 450, 441, 16);
568 }
569 {
570     lblInfo2 = new JLabel();
571     tabRecords.add(lblInfo2);
572     lblInfo2.setText("For exits, the 'Document' column "
573                     + "gives the number of the Item Exit Voucher");
574     lblInfo2.setFont(new java.awt.Font("Dialog",0,10));
575     lblInfo2.setBounds(44, 463, 424, 16);
576 }
```

```
577 {  
578     lblLookFrom = new JLabel();  
579     tabRecords.add(lblLookFrom);  
580     lblLookFrom.setText("From:");  
581     lblLookFrom.setBounds(47, 40, 32, 16);  
582 }  
583 {  
584     ComboBoxModel cmbDayModel = new DefaultComboBoxModel(getArrayOfDays());  
585     cmbDay = new JComboBox();  
586     tabRecords.add(cmbDay);  
587     cmbDay.setModel(cmbDayModel);  
588     cmbDay.setBounds(94, 40, 45, 25);  
589 }  
590 {  
591     ComboBoxModel cmbMonthModel = new DefaultComboBoxModel(getArrayOfMonths());  
592     cmbMonth = new JComboBox();  
593     tabRecords.add(cmbMonth);  
594     cmbMonth.setModel(cmbMonthModel);  
595     cmbMonth.setBounds(151, 40, 88, 25);  
596 }  
597 {  
598     ComboBoxModel cmbYearModel = new DefaultComboBoxModel(getArrayOfYears());  
599     cmbYear = new JComboBox();  
600     tabRecords.add(cmbYear);  
601     cmbYear.setModel(cmbYearModel);  
602     cmbYear.setBounds(251, 40, 63, 25);  
603 }  
604 {  
605     lblLookUntil = new JLabel();  
606     tabRecords.add(lblLookUntil);  
607     lblLookUntil.setText("Until:");  
608     lblLookUntil.setBounds(51, 78, 28, 16);  
609 }  
610 {  
611     ComboBoxModel cmbDayUntilModel = new DefaultComboBoxModel(getArrayOfDays());  
612     cmbDayUntil = new JComboBox();
```

```
613     tabRecords.add(cmbDayUntil);
614     cmbDayUntil.setModel(cmbDayUntilModel);
615     cmbDayUntil.setBounds(94, 78, 45, 25);
616     cmbDayUntil.setSelectedIndex(currentDate.get(Calendar.DAY_OF_MONTH)-1);
617 }
618 {
619     ComboBoxModel cmbMonthUntilModel = new DefaultComboBoxModel(getArrayOfMonths());
620     cmbMonthUntil = new JComboBox();
621     tabRecords.add(cmbMonthUntil);
622     cmbMonthUntil.setModel(cmbMonthUntilModel);
623     cmbMonthUntil.setBounds(151, 78, 88, 25);
624     cmbMonthUntil.setSelectedIndex(currentDate.get(Calendar.MONTH));
625 }
626 {
627     ComboBoxModel cmbYearUntilModel = new DefaultComboBoxModel(getArrayOfYears());
628     cmbYearUntil = new JComboBox();
629     tabRecords.add(cmbYearUntil);
630     cmbYearUntil.setModel(cmbYearUntilModel);
631     cmbYearUntil.setBounds(251, 78, 63, 25);
632     cmbYearUntil.setSelectedIndex(
633         currentDate.get(Calendar.YEAR)%ApplicationConstants.TRANSACTION_YEAR_START);
634 }
635 {
636     btnSearch = new JButton();
637     tabRecords.add(btnSearch);
638     btnSearch.setText("Search");
639     btnSearch.setBounds(377, 77, 75, 26);
640     btnSearch.addActionListener(new ActionListener() {
641         public void actionPerformed (ActionEvent e){
642             displayTransactions();
643         }
644     });
645 }
646 {
647     spRecords = new JSeparator();
648     tabRecords.add(spRecords);
```

```
649         spRecords.setBounds(12, 122, 493, 10);
650     }
651     {
652         //Only an admin can delete a transaction
653         if(user.isAdmin()){
654             btnDeleteTransaction = new JButton();
655             btnDeleteTransaction.setText("Delete Record");
656             btnDeleteTransaction.setBounds(353, 386, 122, 26);
657             btnDeleteTransaction.setEnabled(false);
658             tabRecords.add(btnDeleteTransaction);
659             btnDeleteTransaction.addActionListener(new ActionListener() {
660                 public void actionPerformed (ActionEvent e){
661                     performTransactionDelete();
662                 }
663             });
664         }
665     }
666 }
667 pack();
668 this.setSize(535, 561);
669 } catch (Exception e) {
670     e.printStackTrace();
671 }
672 }
673 }
674 }
675 /**
676 * Adaptation of Jigloo-generated method that initializes the GUI and components
677 * Only displays the item image
678 */
679
680 private void initGUIOnlyImage() {
681     try {
682         this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
683
684         String title = item.getName();
```

```
685
686     if(title.length() >= 40){
687         title = title.substring(0, 40) + "...";
688     }
689
690     this.setTitle(title);
691     this.setIconImage(
692         new ImageIcon(getClass().getClassLoader().getResource(
693             ApplicationConstants.WAREHOUSE_LOGO)).getImage());
694     getContentPane().setLayout(null);
695     this.setResizable(false);
696     {
697         tabContainer = new JTabbedPane();
698         getContentPane().add(tabContainer, "Center");
699         tabContainer.setBounds(0, 0, 522, 527);
700     }
701     tabImage = new JPanel();
702     tabContainer.addTab("Image", tabImage);
703     tabImage.setLayout(null);
704     {
705         lblImage = new JLabel();
706         tabImage.add(lblImage);
707         lblImage.setBounds(103, 52, 320, 240);
708         lblImage.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
709         try {
710             lblImage.setIcon(
711                 new ImageIcon(getClass().getClassLoader().getResource(
712                     "Files/Item_Images/" + item.getID() + ".jpg")));
713         } catch (Exception e) {
714             //Item has no Image
715             lblImage.setIcon(
716                 new ImageIcon(getClass().getClassLoader().getResource(
717                     "Files/Item_Images/NoImage.jpg")));
718         }
719     }
720 }
```

```
721         lblItemImage = new JLabel();
722         tabImage.add(lblItemImage);
723         lblItemImage.setText("Item Image");
724         lblItemImage.setFont(new java.awt.Font("Dialog",1,14));
725         lblItemImage.setBounds(12, 18, 493, 16);
726         lblItemImage.setAlignmentX(0.5f);
727         lblItemImage.setHorizontalAlignment(SwingConstants.CENTER);
728     }
729     {
730         spImage = new JSeparator();
731         tabImage.add(spImage);
732         spImage.setBounds(12, 324, 493, 10);
733     }
734     }
735     }
736     pack();
737     this.setSize(535, 561);
738 } catch (Exception e) {
739     e.printStackTrace();
740 }
741 }
742
743 private JTextField getTxtGroup() {
744     if(txtGroup == null) {
745         txtGroup = new JTextField();
746         txtGroup.setBounds(65, 172, 183, 20);
747         txtGroup.setEditable(false);
748         txtGroup.setSize(183, 24);
749     }
750     return txtGroup;
751 }
752
753 private JTextField getJTextField1() {
754     if(txtUM == null) {
755         txtUM = new JTextField();
756         txtUM.setBounds(309, 173, 194, 20);
```

```
757         txtUM.setEditable(false);
758         txtUM.setSize(194, 24);
759     }
760     return txtUM;
761 }
762 /**
763 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
764 */
765
766 /**
767 * Loads the fields with the appropriate data
768 */
770 public void loadFields(){
771     txtCode.setText(" " + item.getCode());
772     txtName.setText(item.getName());
773     txtDescription.setText(item.getDescription());
774     cmbAisle.setSelectedIndex(item.getLocation().getAisle()-1);
775     cmbColumn.setSelectedIndex(item.getLocation().getColumn()-1);
776     cmbRow.setSelectedIndex(item.getLocation().getRow()-'A');
777     txtLocation.setText(item.getLocation().toString());
778     txtCreatedBy.setText(item.getCreatedBy());
779     int day = item.getCreationDate().get(Calendar.DAY_OF_MONTH);
780     int month = item.getCreationDate().get(Calendar.MONTH);
781     int year = item.getCreationDate().get(Calendar.YEAR);
782     String date = day + "/" + month + "/" + year;
783     txtDate.setText(date);
784     cmbGroup.setSelectedItem(item.getGroup());
785     cmbUM.setSelectedItem(item.getUm());
786     txtGroup.setText(item.getGroup().toString());
787     txtUM.setText(item.getUm().toString());
788 }
789 /**
790 * Gets an Item object from the updated fields. Some fields do not change
791 * @return an Item object from the updated fields
792 */
```

```
793     */
794     public Item getItemFromUpdatedFields(){
795         int ID = item.getID();
796         String code = txtCode.getText();
797         String name = txtName.getText();
798         Group group = ((Group)cmbGroup.getSelectedItem());
799         UM um = ((UM)cmbUM.getSelectedItem());
800         String description = txtDescription.getText();
801         String createdBy = item.getCreatedBy();
802         GregorianCalendar creationDate = item.getCreationDate();
803         Location location = new Location(
804             (Integer)cmbAisle.getSelectedItem().byteValue(),
805             ((Integer)cmbColumn.getSelectedItem().byteValue(),
806             ((String)cmbRow.getSelectedItem()).charAt(0)
807         );
808
809         Item item = new Item(ID, code, name, group, um, description, location, createdBy, creationDate);
810
811         return item;
812     }
813
814 /**
815 * Updates an Item in the Items Random Access File and indexes
816 * @param oldCode - the previous code
817 * @param oldName - the previous name
818 * @param item - the item to be updated
819 */
820 public void updateItem(String oldCode, String oldName, Item item){
821     try{
822         RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
823         itemsFile.seek(item.getID() * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
824         item.writeItemToFile(itemsFile);
825         updateCodeIndexFile(oldCode, item);
826         updateNameIndex(oldName, item);
827
828     /*
```

```
829         * Delete indexes from BTREE
830         * Add new Indexes
831         */
832         MainScreen.codeIndexTree.deleteIndex(MainScreen.codeIndexTree.getNode(oldCode));
833         MainScreen.nameIndexTree.deleteIndex(MainScreen.nameIndexTree.getNode(oldName));
834
835         MainScreen.codeIndexTree.insertIndex(new IndexTNode(new Index(item.getID(), item.getCode())));
836         MainScreen.nameIndexTree.insertIndex(new IndexTNode(new Index(item.getID(), item.getName())));
837     } catch (Exception e){
838
839     }
840 }
841
842 /**
843 * Deletes an Item from the Items Random Access File and indexes
844 * @param item - the item to delete
845 * @mastery method achieves HL mastery factor 2 by deleting an Item from the Items Random Access File
846 * by manipulating the file pointer using the seek method and flagging the item as deleted
847 */
848 public void deleteItem(Item item){
849     try{
850         RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
851         itemsFile.seek(item.getID() * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
852         itemsFile.writeInt(-999);
853         deleteCodeIndex(item);
854         deleteNameIndex(item);
855         MainScreen.codeIndexTree.deleteIndex(MainScreen.codeIndexTree.getNode(item.getCode()));
856         MainScreen.nameIndexTree.deleteIndex(MainScreen.nameIndexTree.getNode(item.getName()));
857     } catch (Exception e){
858
859     }
860 }
861
862 /**
863 * Updates an Item in the Index file by name
864 * @param oldCode - the previous name
```

```
865     * @param item - the item to update
866     */
867     public void updateNameIndex(String oldName, Item item){
868         try{
869             File file = new File(ApplicationConstants.NAME_INDEX_FILE);
870             File tmpFile = new File(ApplicationConstants.TMP_NAME_INDEX_FILE);      //temporary file
871             FileReader reader = new FileReader(file);
872             BufferedReader buff = new BufferedReader(reader);
873             FileWriter writer = new FileWriter(tmpFile);
874             BufferedWriter buffwriter = new BufferedWriter(writer);
875
876             boolean eof = false;          //stores if the end of the file (eof) has been reached
877
878             while(!eof){
879                 String line = buff.readLine();
880                 if(line == null){
881                     eof = true;           //the end of the file has been reached
882                 } else {
883                     if(line.equals(oldName + " | " + item.getID())){
884                         buffwriter.write(" " + new Index(item.getID(), item.getName()).toString());
885                         buffwriter.newLine();
886                     } else {
887                         buffwriter.write(line);
888                         buffwriter.newLine();
889                     }
890                 }
891             }
892
893             buffwriter.close();
894             buff.close();
895
896             //Rename temporary file to old file
897             file.delete();
898             tmpFile.renameTo(file);
899
900         } catch (Exception e){
```

```
901
902     }
903 }
904
905 /**
906 * Updates an Item in the Index file by code
907 * @param oldCode - the previous code
908 * @param item - the item to update
909 */
910 public void updateCodeIndexFile(String oldCode, Item item){
911     try{
912         File file = new File(ApplicationConstants.CODE_INDEX_FILE);
913         File tmpFile = new File(ApplicationConstants.TMP_CODE_INDEX_FILE);      //temporary file
914         FileReader reader = new FileReader(file);
915         BufferedReader buff = new BufferedReader(reader);
916         FileWriter writer = new FileWriter(tmpFile);
917         BufferedWriter buffwriter = new BufferedWriter(writer);
918
919         boolean eof = false;           //stores if the end of the file (eof) has been reached
920
921         while(!eof){
922             String line = buff.readLine();
923             if(line == null){
924                 eof = true;          //the end of the file has been reached
925             } else {
926                 if(line.equals(oldCode + " | " + item.getID())){
927                     buffwriter.write(" " + new Index(item.getID(), item.getCode()).toString());
928                     buffwriter.newLine();
929                 } else {
930                     buffwriter.write(line);
931                     buffwriter.newLine();
932                 }
933             }
934         }
935
936         buffwriter.close();
```

```
937         buff.close();
938
939         //Rename temporary file to old file
940         file.delete();
941         tmpFile.renameTo(file);
942
943     } catch (Exception e){
944
945     }
946 }
947
948 /**
949 * Deletes an Item from the Index file by name
950 * @param item - the item to delete
951 */
952 public void deleteNameIndex(Item item){
953     try{
954         File file = new File(ApplicationConstants.NAME_INDEX_FILE);
955         File tmpFile = new File(ApplicationConstants.TMP_NAME_INDEX_FILE);      //the temporary file
956         FileReader reader = new FileReader(file);
957         BufferedReader buff = new BufferedReader(reader);
958         FileWriter writer = new FileWriter(tmpFile);
959         BufferedWriter buffwriter = new BufferedWriter(writer);
960
961         boolean eof = false;           //stores if the end of the file (eof) has been reached
962
963         while(!eof){
964             String line = buff.readLine();
965             if(line == null){
966                 eof = true;          //the end of the file has been reached
967             } else {
968                 if(!line.equals(item.getName() + " | " + item.getID())) {
969                     buffwriter.write(line);
970                     buffwriter.newLine();
971                 }
972             }
973         }
974     } catch (Exception e){
975         System.out.println("Error deleting item from index file");
976     }
977 }
```

```
973     }
974
975     buffwriter.close();
976     buff.close();
977
978     //Rename temporary file to old file
979     file.delete();
980     tmpFile.renameTo(file);
981
982 } catch (Exception e){
983
984 }
985 }
986
987 /**
988 * Deletes an Item from the Index file by code
989 * @param item - the item to delete
990 * @mastery achieves HL mastery factor 18 by deleting an index by code from the CodeIndex sequential
991 * file without reading the entire file into RAM
992 */
993 public void deleteCodeIndex(Item item){
994     try{
995         File file = new File(ApplicationConstants.CODE_INDEX_FILE);
996         File tmpFile = new File(ApplicationConstants.TMP_CODE_INDEX_FILE);      //the temporary file
997         FileReader reader = new FileReader(file);
998         BufferedReader buff = new BufferedReader(reader);
999         FileWriter writer = new FileWriter(tmpFile);
1000        BufferedWriter buffwriter = new BufferedWriter(writer);
1001
1002        boolean eof = false;           //stores if the end of the file (eof) has been reached
1003
1004        while(!eof){
1005            String line = buff.readLine();
1006            if(line == null){
1007                eof = true;           //the end of the file has been reached
1008            } else {
```

```
1009         if(!line.equals(item.getCode() + " | " + item.getID())) {
1010             buffwriter.write(line);
1011             buffwriter.newLine();
1012         }
1013     }
1014 }
1015
1016 buffwriter.close();
1017 buff.close();
1018
1019 //Rename temporary file to old file
1020 file.delete();
1021 tmpFile.renameTo(file);
1022
1023 } catch (Exception e){
1024
1025 }
1026 }
1027
1028 /**
1029 * Asks the user for confirmation, and if so deletes the item
1030 */
1031 public void performDelete(){
1032     int userResponse = JOptionPane.showConfirmDialog(
1033         this, "The item will be permanently deleted. Continue?", "Delete Item",
1034         JOptionPane.YES_NO_OPTION,JOptionPane.WARNING_MESSAGE);
1035     if(userResponse == JOptionPane.YES_OPTION){
1036         deleteItem(item);
1037         try {
1038             File transactions = new File(
1039                 ApplicationConstants.TRANSACTIONS_FOLDER + item.getID() + ".txt");
1040             transactions.delete();
1041             File image = new File(ApplicationConstants.ITEM_IMAGES_FOLDER + item.getID() + ".jpg");
1042             image.delete();
1043         } catch (Exception e) {
1044             //One of the files may not exist
```

```
1045     }
1046     JOptionPane.showMessageDialog(
1047         this, "The item has been deleted", "Success", JOptionPane.INFORMATION_MESSAGE);
1048     this.dispose();
1049     ChangeScreen.setBlankScreen(user);
1050 }
1051 }
1052 /**
1053 * Validates the user input
1054 * @return true if input is valid, else false
1055 */
1056 public boolean isCorrectInput(){
1057
1058     if(!txtName.getText().equals(""))
1059         || txtName.getText() != null && txtCode.getText().length() == 6){
1060         return true;
1061     } else {
1062         return false;
1063     }
1064 }
1065 }
1066 /**
1067 * Enables GUI components for editing
1068 */
1069 public void turnEditingOn() {
1070     btnEditing.setText("Save Changes");
1071     txtCode.setEditable(true);
1072     lblCodeInfo.setVisible(true);
1073     txtName.setEditable(true);
1074     txtDescription.setEditable(true);
1075     cmbUM.setVisible(true);
1076     cmbGroup.setVisible(true);
1077     txtUM.setVisible(false);
1078     txtGroup.setVisible(false);
1079 }
```

```
1081     txtLocation.setVisible(false);
1082     cmbAisle.setVisible(true);
1083     lblAisle.setVisible(true);
1084     cmbRow.setVisible(true);
1085     lblRow.setVisible(true);
1086     cmbColumn.setVisible(true);
1087     lblColumn.setVisible(true);
1088     lblRequired.setVisible(true);
1089     lblNameInfo.setVisible(true);
1090     lblOptional.setVisible(true);
1091     lblDescriptionInfo.setVisible(true);
1092 }
1093 /**
1094 * Disables GUI components for only viewing
1095 */
1096 public void turnEditingOff(){
1097     btnEditing.setText("Turn Editing On");
1098     txtCode.setEditable(false);
1099     lblCodeInfo.setVisible(false);
1100     txtName.setEditable(false);
1101     txtDescription.setEditable(false);
1102     cmbUM.setVisible(false);
1103     cmbGroup.setVisible(false);
1104     txtUM.setVisible(true);
1105     txtGroup.setVisible(true);
1106     txtLocation.setVisible(true);
1107     txtLocation.setEnabled(true);
1108     txtLocation.setEditable(false);
1109     cmbAisle.setVisible(false);
1110     lblAisle.setVisible(false);
1111     cmbRow.setVisible(false);
1112     lblRow.setVisible(false);
1113     cmbColumn.setVisible(false);
1114     lblColumn.setVisible(false);
1115     lblRequired.setVisible(false);
```

```
1117     lblNameInfo.setVisible(false);
1118     lblOptional.setVisible(false);
1119     lblDescriptionInfo.setVisible(false);
1120 }
1121 /**
1122 * Enables or disabled GUI components for editing or viewing
1123 */
1124 public void enableEditing(){
1125     boolean isEditable = txtName.isEditable();
1126
1127     if(!isEditable){
1128         turnEditingOn();
1129         this.validate();
1130         tabDescription.validate();
1131         this.repaint();
1132         tabDescription.repaint();
1133     } else {
1134         if(isCompleteInput()){
1135             turnEditingOff();
1136             int userResponse = JOptionPane.showConfirmDialog(
1137                 this, "This will permanently update the \nitem. Continue?", "Update Item",
1138                 JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
1139
1140             if(userResponse == JOptionPane.YES_OPTION){
1141                 byte aisle = ((Integer) cmbAisle.getSelectedItem()).byteValue();
1142                 char row = ((String) cmbRow.getSelectedItem()).charAt(0);
1143                 byte column = ((Integer) cmbColumn.getSelectedItem()).byteValue();
1144                 Location location = new Location(aisle, column, row);
1145                 ArrayList itemsInLocation = Items.searchByLocation(location);
1146
1147                 for(int i=0;i<itemsInLocation.size();i++){
1148                     if(((Item) itemsInLocation.get(i)).getID() == item.getID()){
1149                         itemsInLocation.remove(i);
1150                     }
1151                 }
1152             }
1153         }
1154     }
1155 }
```

```
1153
1154     if(itemsInLocation == null || itemsInLocation.isEmpty()){
1155         updateItem(item.getCode(), item.getName(), getItemFromUpdatedFields());
1156         this.item = getItemFromUpdatedFields();
1157         loadFields();
1158         this.validate();
1159         tabDescription.validate();
1160         this.repaint();
1161         tabDescription.repaint();
1162         JOptionPane.showMessageDialog(
1163             this, "The item has been updated", "Success", JOptionPane.INFORMATION_MESSAGE);
1164     } else {
1165         String items = "The following items are stored in the location " + location.toString()
1166             + ": \n";
1167         for(int i=0;i<itemsInLocation.size();i++){
1168             Item item = (Item)itemsInLocation.get(i);
1169             String itemName = item.getName();
1170             if(itemName.length()>40){
1171                 itemName = itemName.substring(0, 19) + "...";
1172             }
1173             items = items + "\n" + (i+1) + ". " + itemName;
1174         }
1175
1176         items = items + "\n" + "\n" + "Assign this location to the new item anyway?";
1177
1178         int userResponseLocation = JOptionPane.showConfirmDialog(
1179             this, items, "Location Taken",
1180             JOptionPane.YES_NO_OPTION);
1181         if(userResponseLocation == JOptionPane.YES_OPTION){
1182             updateItem(item.getCode(), item.getName(), getItemFromUpdatedFields());
1183             this.item = getItemFromUpdatedFields();
1184             loadFields();
1185             this.validate();
1186             tabDescription.validate();
1187             this.repaint();
1188             tabDescription.repaint();
```

```
1189         JOptionPane.showMessageDialog(
1190             this, "The item has been updated", "Success", JOptionPane.INFORMATION_MESSAGE);
1191     } else {
1192         loadFields();
1193         JOptionPane.showMessageDialog(
1194             this, "The item has not been updated", "No change", JOptionPane.ERROR_MESSAGE);
1195     }
1196 }
1197 } else {
1198     loadFields();
1199     JOptionPane.showMessageDialog(
1200         this, "The item has not been updated", "No change", JOptionPane.ERROR_MESSAGE);
1201     }
1202 } else {
1203     JOptionPane.showMessageDialog(
1204         this, "Some fields are missing or incorrect", "Error", JOptionPane.ERROR_MESSAGE);
1205     }
1206 }
1207 }
1208 /**
1209 * Populates the transactions list with all transactions from the file
1210 */
1211 public void populateList(){
1212     list = new TransactionList();
1213
1214     try{
1215         File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + item.getID() + ".txt");
1216         FileReader reader = new FileReader(file);
1217         BufferedReader buff = new BufferedReader(reader);
1218
1219         boolean eof = false; //stores whether the end of the file has been reached
1220
1221         while(!eof){
1222             String line = buff.readLine();
```

```
1225
1226         if(line == null){
1227             eof = true;
1228         } else {
1229             list.addToHead(new TransactionNode(Transactions.readTransaction(item, line)));
1230         }
1231     }
1232     buff.close();
1233 } catch (Exception e){
1234     JOptionPane.showMessageDialog(
1235         MainScreen.contentPane, "One or files cannot be accessed. \n"
1236         + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
1237 }
1238
1239 }
1240
1241 /**
1242 * Gets a String array of the days in a month
1243 * @return a String array of the days in a month
1244 */
1245 public String[] getArrayOfDays(){
1246     String[] days = new String[31];
1247
1248     for (int i=0;i<days.length;i++){
1249         days[i] = (" " + (i+1));
1250     }
1251
1252     return days;
1253 }
1254
1255 /**
1256 * Gets a String array of the months in a year
1257 * @return a String array of the names of months in a year
1258 */
1259 public String[] getArrayOfMonths(){
1260     String[] months = new String[12];
```

```
1261     months[0] = "January";
1262     months[1] = "February";
1263     months[2] = "March";
1264     months[3] = "April";
1265     months[4] = "May";
1266     months[5] = "June";
1267     months[6] = "July";
1268     months[7] = "August";
1269     months[8] = "September";
1270     months[9] = "October";
1271     months[10] = "November";
1272     months[11] = "December";
1273
1274     return months;
1275 }
1276
1277 /**
1278 * Gets a String array of the years from when the transactions start to be recorded to the current year
1279 * @return a String array of year numbers
1280 */
1281
1282 public String[] getArrayOfYears(){
1283     GregorianCalendar currentDate = new GregorianCalendar();
1284     int currentYear = currentDate.get(Calendar.YEAR);
1285     int startYear = ApplicationConstants.TRANSACTION_YEAR_START;
1286     String[] years = new String[currentYear - startYear + 1];
1287
1288     for(int i=0;i<years.length;i++){
1289         years[i] = (" " + (startYear));
1290         startYear = startYear+1;
1291     }
1292
1293     return years;
1294 }
1295
1296 }
```

```
1297 /**
1298  * Sets the width of the Transactions table columns
1299 */
1300 public void setColumnWidths(){
1301     //Set Column "Date" width
1302     int vColIndex = 0;
1303     TableColumn colDate = tblTransactions.getColumnModel().getColumn(vColIndex);
1304     int width = 100;
1305     colDate.setPreferredWidth(width);
1306
1307     //Set Column "Type" width
1308     TableColumn colType = tblTransactions.getColumnModel().getColumn(1);
1309     colType.setPreferredWidth(70);
1310
1311     //Set Column "Quantity" width
1312     TableColumn colQuantity = tblTransactions.getColumnModel().getColumn(2);
1313     colQuantity.setPreferredWidth(70);
1314
1315     //Set Column "Balance" width
1316     TableColumn colBalance = tblTransactions.getColumnModel().getColumn(3);
1317     colBalance.setPreferredWidth(74);
1318
1319     //Set Column "Document" width
1320     TableColumn colDocument = tblTransactions.getColumnModel().getColumn(4);
1321     colDocument.setPreferredWidth(114);
1322
1323     //Set Column "Delivered By" width
1324     TableColumn colDeliveredBy = tblTransactions.getColumnModel().getColumn(5);
1325     colDeliveredBy.setPreferredWidth(140);
1326
1327     //Set Column "Delivered By" width
1328     TableColumn colRequestedBy = tblTransactions.getColumnModel().getColumn(6);
1329     colRequestedBy.setPreferredWidth(144);
1330
1331     //Set Column "Delivered By" width
1332     TableColumn colStaff = tblTransactions.getColumnModel().getColumn(7);
```

```
1333     colStaff.setPreferredWidth(143);
1334 }
1335
1336 /**
1337 * Displays transactions in the Transactions table from the date range that the user chooses
1338 */
1339 public void displayTransactions(){
1340     //Start date
1341     int dayStart = new Integer((String)cmbDay.getSelectedItem()).intValue();
1342     int monthStart = cmbMonth.getSelectedIndex()+1;
1343     int yearStart = new Integer((String)cmbYear.getSelectedItem()).intValue();
1344     GregorianCalendar dateFrom = new GregorianCalendar(yearStart, monthStart, dayStart);
1345
1346     //End date
1347     int dayEnd = new Integer((String)cmbDayUntil.getSelectedItem()).intValue();
1348     int monthEnd = cmbMonthUntil.getSelectedIndex()+1;
1349     int yearEnd = new Integer((String)cmbYearUntil.getSelectedItem()).intValue();
1350     GregorianCalendar dateTo = new GregorianCalendar(yearEnd, monthEnd, dayEnd);
1351
1352     if(dateFrom.compareTo(dateTo)<=0){
1353         ArrayList transactionsToDisplay = list.searchTransactions(dateFrom, dateTo);
1354         TransactionsTableModel tblTransactionsModel = new TransactionsTableModel(transactionsToDisplay);
1355         tblTransactions.setModel(tblTransactionsModel);
1356         tblTransactions.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
1357         setColumnWidths();
1358         tblTransactions.validate();
1359     } else {
1360         JOptionPane.showMessageDialog(
1361             this, "You have entered an invalid date range", "Error", JOptionPane.ERROR_MESSAGE);
1362     }
1363 }
1364
1365 /**
1366 * Deletes a Transaction
1367 */
1368 public void performTransactionDelete(){
```

```
1369     int userResponse = JOptionPane.showConfirmDialog(
1370         this, "This transaction will be permanently \\ndeleted. Continue?", 
1371         "Delete Transaction", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);
1372     if(userResponse == JOptionPane.YES_OPTION){
1373         int row = tblTransactions.getSelectedRow();
1374         TransactionsTableModel model = (TransactionsTableModel)tblTransactions.getModel();
1375         TransactionRecord transactionToDelete =
1376             (TransactionRecord) model.getTransactionsToDisplay().get(row);      // transaction to delete
1377         deleteTransactionFromFile(transactionToDelete);
1378         list.deleteTransaction(list.getNode(transactionToDelete.getID()));
1379
1380         //Update table
1381         ArrayList transactionsToDisplay = list.searchTransactions(dateStart, dateEnd);
1382         TransactionsTableModel tblTransactionsModel =
1383             new TransactionsTableModel(transactionsToDisplay);
1384         tblTransactions.setModel(tblTransactionsModel);
1385         tblTransactions.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
1386         setColumnWidths();
1387         tblTransactions.validate();
1388         JOptionPane.showMessageDialog(
1389             this, "The transaction has been deleted", "Success", JOptionPane.INFORMATION_MESSAGE);
1390     }
1391
1392 }
1393
1394 /**
1395 * Deletes a TransactionRecord from the Transactions file for this item
1396 * @param transactionToDelete - the transaction to delete
1397 */
1398 public void deleteTransactionFromFile(TransactionRecord transactionToDelete){
1399     try{
1400         File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + this.item.getID() + ".txt");
1401         File tmpFile = new File(
1402             ApplicationConstants.TRANSACTIONS_FOLDER + this.item.getID() + "_tmp" + ".txt");
1403         FileReader reader = new FileReader(file);
1404         BufferedReader buff = new BufferedReader(reader);
```

```
1405     FileWriter writer = new FileWriter(tmpFile);
1406     BufferedWriter buffwriter = new BufferedWriter(writer);
1407
1408     boolean eof = false;           //stores if the end of the file (eof) has been reached
1409
1410     while(!eof){
1411         String line = buff.readLine();
1412         if(line == null){
1413             eof = true;           //the end of the file has been reached
1414         } else {
1415             TransactionRecord currentTransaction = Transactions.readTransaction(this.item, line);
1416             if(currentTransaction.getID() != transactionToDelete.getID()) {
1417                 buffwriter.write(line);
1418                 buffwriter.newLine();
1419             }
1420         }
1421     }
1422
1423     buffwriter.close();
1424     buff.close();
1425
1426     //Rename temporary file to old file
1427     file.delete();
1428     tmpFile.renameTo(file);
1429
1430 } catch (Exception e){
1431
1432 }
1433
1434 /**
1435 * Enables the delete transaction button
1436 * @param i - the value from a row selection (-1 if no row is selected)
1437 */
1438
1439 public void enableDeleteTransaction(int i){
1440     if(i<0){
```

```
1441     btnDeleteTransaction.setEnabled(false);
1442 } else {
1443     btnDeleteTransaction.setEnabled(true);
1444 }
1445 }
1446 /**
1447 * Validates the user input
1448 * @return true if input is valid, else false
1449 */
1450 public boolean isCompleteInput(){
1451
1452     if(!txtName.getText().equals("") || txtName.getText() != null
1453         && txtCode.getText().length() == 6){
1454         return true;
1455     } else {
1456         return false;
1457     }
1458 }
1459 }
1460 /**
1461 * Gets an image from the File that the user selected and displays it in the Item Image box
1462 * @return the new image file, or null if no change
1463 */
1464 public File getAndDisplayImage(){
1465     //Get and Display a File Chooser
1466     JFileChooser fileChooser = new JFileChooser();
1467     fileChooser.setAcceptAllFileFilterUsed(false);
1468     fileChooser.setFileFilter(new JpgFilter());
1469     fileChooser.setDialogTitle("Choose an item image");
1470     int returnValue = fileChooser.showOpenDialog(this);
1471
1472     if (returnValue == JFileChooser.APPROVE_OPTION) {
1473         if(fileChooser.getSelectedFile() != null){
1474             File file = fileChooser.getSelectedFile();
```

```
1477     try{
1478         Image image = ImageIO.read(file);
1479         lblImage.setIcon(new ImageIcon(image));
1480         return file;
1481     } catch (Exception e){
1482
1483     }
1484 }
1485
1486
1487     return null;
1488
1489 }
1490
1491 /**
1492 * Changes an item's image
1493 */
1494 public void changeItemImage(){
1495
1496     File file = getAndDisplayImage();
1497     if(file!=null){
1498         try{
1499             File oldImage = new File(ApplicationConstants.ITEM_IMAGES_FOLDER + item.getID() + ".jpg");
1500             oldImage.exists();
1501             oldImage.delete();
1502         } catch (Exception e){
1503             //if the file doesn't exist
1504             e.printStackTrace();
1505         }
1506
1507         try{
1508             File newImageFile = copyFile(file, ApplicationConstants.ITEM_IMAGES_FOLDER);
1509             newImageFile.renameTo(new File(ApplicationConstants.ITEM_IMAGES_FOLDER
1510                 + item.getID() + ".jpg"));
1511         } catch (Exception e){
1512             e.printStackTrace();
```

```
1513         }
1514     }
1515 }
1516 }
1517 /**
1518 * Copies a file to a destination
1519 * @author Vinod Singh, with modifications by Alvaro Morales
1520 * @authorswebsite http://blog.vinodsingh.com/2009/06/copy-move-and-delete-files-using-java.html
1521 * @date 12 June 2009
1522 * @param file - the file to copy
1523 * @param destination - the destination path of the file
1524 * @return the file that has been copied
1525 * @throws IOException if file accessing does not work
1526 */
1527 public File copyFile(File file, String destination) throws IOException {
1528     FileChannel in = null;
1529     FileChannel out = null;
1530     try {
1531         in = new FileInputStream(file).getChannel();
1532         File outFile = new File(destination, file.getName());
1533         out = new FileOutputStream(outFile).getChannel();
1534         in.transferTo(0, in.size(), out);
1535         return outFile;
1536     } finally {
1537         if (in != null)
1538             in.close();
1539         if (out != null)
1540             out.close();
1541     }
1542 }
1543 }
1544 }
1545 }
1546 }
```

```
1 import java.awt.event.*;
2 import java.util.*;
3 import javax.swing.*;
4 import javax.swing.table.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * The Item Entry screen to process item entry transactions
10 * @author Alvaro Morales
11 * @date 13/07/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class ItemEntry extends javax.swing.JPanel {
18
19 /**
20 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
21 */
22
23 /**
24 * GUI components generated by Jigloo
25 */
26
27 private JLabel lblItemEntry;
28 private JTable tblQuantities;
29 private JLabel lblQuantitiesInfo;
30 private JLabel lblQuantities;
31 private JScrollPane scpQuantities;
32 private JButton btnFinish;
33 private JSeparator spEntryList;
34 private JTextField txtDeliveredBy;
35 private JLabel lblRequestedBy;
36 private JTextField txtDate;
```

```
37     private JLabel lblDate;
38     private JTextField txtReceivedBy;
39     private JLabel lblDispatchedBy;
40     private JTextField txtProof;
41     private JLabel lblRequiredFields;
42     private JLabel lblItemExitVoucher;
43
44     /**
45      * An ArrayList of items to process
46      */
47     private ArrayList itemsToProcess;
48
49     /**
50      * The user that logged in
51      */
52     private User user;
53
54     /**
55      * The current date
56      */
57     private GregorianCalendar date;
58
59
60     {
61         //Set Look & Feel
62         try {
63             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
64         } catch(Exception e) {
65             e.printStackTrace();
66         }
67     }
68
69     /**
70      * Constructs a new ItemEntry object
71      * @param user - the user that logged in
72      * @param itemsToProcess - an ArrayList of items to process
```

```
73     */
74     public ItemEntry(User user, ArrayList itemsToProcess) {
75         super();
76         this.user = user;
77         date = new GregorianCalendar();
78         this.itemsToProcess = itemsToProcess;
79         initGUI();
80     }
81
82     /**
83      * Method generated by Jigloo to initialize GUI components
84      */
85     private void initGUI() {
86         try {
87             this.setPreferredSize(new java.awt.Dimension(800, 500));
88             this.setLayout(null);
89             {
90                 lblItemEntry = new JLabel();
91                 this.add(lblItemEntry);
92                 lblItemEntry.setText("4. Item Entry");
93                 lblItemEntry.setFont(new java.awt.Font("Dialog",1,14));
94                 lblItemEntry.setBounds(12, 12, 96, 16);
95             }
96             {
97                 lblItemExitVoucher = new JLabel();
98                 this.add(lblItemExitVoucher);
99                 lblItemExitVoucher.setText("Proof of Reception No.* :");
100                lblItemExitVoucher.setBounds(36, 56, 132, 16);
101                lblItemExitVoucher.setSize(137, 16);
102            }
103            {
104                lblRequiredFields = new JLabel();
105                this.add(lblRequiredFields);
106                lblRequiredFields.setText("Fields marked with an * are required");
107                lblRequiredFields.setFont(new java.awt.Font("Dialog",0,10));
108                lblRequiredFields.setBounds(12, 28, 205, 16);

```

```
109
110
111     txtProof = new JTextField();
112     this.add(txtProof);
113     txtProof.setBounds(180, 54, 188, 20);
114     txtProof.setSize(206, 24);
115     txtProof.setDocument(new TextFieldDigitLimit(8));
116 }
117 {
118     lblDispatchedBy = new JLabel();
119     this.add(lblDispatchedBy);
120     lblDispatchedBy.setText("Received By:");
121     lblDispatchedBy.setBounds(457, 56, 72, 16);
122 }
123 {
124     txtReceivedBy = new JTextField();
125     this.add(txtReceivedBy);
126     txtReceivedBy.setBounds(541, 54, 206, 20);
127     txtReceivedBy.setEditable(false);
128     txtReceivedBy.setSize(206, 24);
129     txtReceivedBy.setText(user.getUsername());
130 }
131 {
132     lblDate = new JLabel();
133     this.add(lblDate);
134     lblDate.setText("Date:");
135     lblDate.setBounds(500, 94, 29, 16);
136 }
137 {
138     txtDate = new JTextField();
139     this.add(txtDate);
140     txtDate.setBounds(541, 90, 206, 24);
141     txtDate.setEditable(false);
142     int day = date.get(Calendar.DAY_OF_MONTH);
143     int month = date.get(Calendar.MONTH);
144     int year = date.get(Calendar.YEAR);
```

```
145         txtDate.setText(day + "/" + month + "/" + year);
146     }
147     {
148         lblRequestedBy = new JLabel();
149         this.add(lblRequestedBy);
150         lblRequestedBy.setText("Delivered By* :");
151         lblRequestedBy.setBounds(91, 94, 82, 16);
152     }
153     {
154         txtDeliveredBy = new JTextField();
155         this.add(txtDeliveredBy);
156         txtDeliveredBy.setBounds(180, 90, 206, 24);
157     }
158     {
159         spEntryList = new JSeparator();
160         this.add(spEntryList);
161         spEntryList.setBounds(12, 127, 776, 10);
162     }
163     {
164         scpQuantities = new JScrollPane();
165         this.add(scpQuantities);
166         scpQuantities.setBounds(36, 188, 725, 256);
167     {
168         TransactionProcessingTableModel tblQuantitiesModel =
169             new TransactionProcessingTableModel(itemsToProcess);
170         tblQuantities = new JTable();
171         scpQuantities.setViewportView(tblQuantities);
172         tblQuantities.setModel(tblQuantitiesModel);
173         tblQuantities.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
174         tblQuantities.addMouseListener(new MouseAdapter() {
175             public void mouseClicked(MouseEvent e) {
176                 if (e.getClickCount() == 2) {
177                     JTable tblResults = (JTable)e.getSource();
178                     int row = tblResults.getSelectedRow();
179                     TransactionProcessingTableModel model =
180                         (TransactionProcessingTableModel)tblQuantities.getModel();
```

```
181         Item item = (Item) model.getSearchResults().get(row);
182         ItemDescription description = new ItemDescription(item);
183         description.setVisible(true);
184     }
185 }
186 );
187
188 //Set Column "Item ID" width
189 int vColIndex = 0;
190 TableColumn colID = tblQuantities.getColumnModel().getColumn(vColIndex);
191 int width = 100;
192 colID.setPreferredWidth(width);
193
194 //Set Column "Item Name" width
195 TableColumn colName = tblQuantities.getColumnModel().getColumn(1);
196 colName.setPreferredWidth(308);
197
198 //Set Column "UM" width
199 TableColumn colUM = tblQuantities.getColumnModel().getColumn(2);
200 colUM.setPreferredWidth(50);
201
202 //Set Column "In Stock" width
203 TableColumn colStock = tblQuantities.getColumnModel().getColumn(3);
204 colStock.setPreferredWidth(100);
205
206 //Set Column "Quantity" width
207 TableColumn colQuantity = tblQuantities.getColumnModel().getColumn(4);
208 colQuantity.setPreferredWidth(150);
209
210 }
211 }
212 {
213     btnFinish = new JButton();
214     this.add(btnFinish);
215     btnFinish.setText("Finish");
216     btnFinish.setBounds(672, 456, 89, 26);
```

```
217     btnFinish.addActionListener(new ActionListener() {
218         public void actionPerformed (ActionEvent e){
219             performTransactionProcessing();
220         }
221     });
222 }
223 {
224     lblQuantities = new JLabel();
225     this.add(lblQuantities);
226     lblQuantities.setText("5. Quantities");
227     lblQuantities.setFont(new java.awt.Font("Dialog",1,14));
228     lblQuantities.setBounds(12, 143, 96, 16);
229 }
230 {
231     lblQuantitiesInfo = new JLabel();
232     this.add(lblQuantitiesInfo);
233     lblQuantitiesInfo.setText("Fill the 'Quantity' column for each item to be entered into stock");
234     lblQuantitiesInfo.setFont(new java.awt.Font("Dialog",0,10));
235     lblQuantitiesInfo.setBounds(12, 160, 631, 16);
236 }
237 } catch (Exception e) {
238     e.printStackTrace();
239 }
240 }
241 /**
242 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
243 */
244
245 /**
246 * Checks that the input is correct, and if so calls the methods to process the transactions
247 */
248
249 public void performTransactionProcessing(){
250     if(isCorrectInput()){
251         processAllTransactions();
252         JOptionPane.showMessageDialog(this, "Transactions successfully processed",
```

```
253         "Success", JOptionPane.INFORMATION_MESSAGE);
254     ChangeScreen.setBlankScreen(user);
255 } else {
256     JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);
257 }
258 }
259 /**
260 * Processes all the transactions
261 */
262 public void processAllTransactions(){
263     TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
264     TransactionRecord[] transactionsToProcess = new TransactionRecord[model.getRowCount()];
265
266     for(int i=0;i<transactionsToProcess.length;i++){
267         transactionsToProcess[i] = new TransactionRecord();
268         transactionsToProcess[i].setType(ApplicationConstants.ITEM_ENTRY);      //screen deals with item entry
269         transactionsToProcess[i].setDocument(txtProof.getText());
270         transactionsToProcess[i].setDate(this.date);
271         transactionsToProcess[i].setWarehouseWorker(txtReceivedBy.getText());
272         transactionsToProcess[i].setThirdParty(txtDeliveredBy.getText());
273         transactionsToProcess[i].setQuantity(((Integer)model.getValueAt(i, 4)).intValue());
274         transactionsToProcess[i].setItem((Item)model.getSearchResults().get(i));
275         transactionsToProcess[i].processTransaction();
276     }
277 }
278 /**
279 * Checks if all the necessary fields have been filled in
280 * @return true if all fields have been filled in, else false
281 */
282 public boolean isCorrectInput(){
283     TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
284     boolean isComplete = true;
285 }
```

```
289     for(int i=0;i<model.getRowCount();i++) {
290         TransactionRecord record = model.getTransactions()[i];
291         if(record.getQuantity() <= 0) {
292             isComplete = false;
293             break;
294         }
295     }
296
297     if(txtProof.getText().equals("") || txtDeliveredBy.getText().equals("") || !isComplete) {
298         return false;
299     } else {
300         return true;
301     }
302 }
303
304 /**
305 * Opens the description of a selected item (row in a table)
306 * @param row - the selected row index
307 */
308 public void openItemDescription(int row){
309     TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
310     Item item = (Item)model.getSearchResults().get(row);
311     ItemDescription description = new ItemDescription(item, user);
312     description.setVisible(true);
313 }
314
315 }
```

```
1 import javax.swing.*;
2 import java.util.*;
3 import java.awt.event.*;
4 import javax.swing.table.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * The Item Exit screen to process item withdrawal transactions
10 * @author Alvaro Morales
11 * @date 13/07/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class ItemExit extends javax.swing.JPanel {
18
19 /**
20 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
21 */
22
23 /**
24 * GUI components generated by Jigloo
25 */
26
27 private JLabel lblItemExit;
28 private JTable tblQuantities;
29 private JLabel lblQuantitiesInfo;
30 private JLabel lblQuantities;
31 private JScrollPane scpQuantities;
32 private JButton btnNext;
33 private JSeparator spExitList;
34 private JTextField txtRequestedBy;
35 private JLabel lblRequestedBy;
36 private JTextField txtDate;
```

```
37     private JLabel lblDate;
38     private JTextField txtDispatchedBy;
39     private JLabel lblDispatchedBy;
40     private JTextField txtVoucher;
41     private JLabel lblRequiredFields;
42     private JLabel lblItemExitVoucher;
43
44     /**
45      * An ArrayList of items to process
46      */
47     private ArrayList itemsToProcess;
48
49     /**
50      * The user that logged in
51      */
52     private User user;
53
54     /**
55      * The current date
56      */
57     private GregorianCalendar date;
58
59
60     {
61         //Set Look & Feel
62         try {
63             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
64         } catch(Exception e) {
65             e.printStackTrace();
66         }
67     }
68
69     /**
70      * Constructs a new ItemExit object
71      * @param user - the user that logged in
72      * @param itemsToProcess - an ArrayList of items to withdraw from the warehouse
```

```
73     */
74     public ItemExit(User user, ArrayList itemsToProcess) {
75         super();
76         this.user = user;
77         date = new GregorianCalendar();
78         this.itemsToProcess = itemsToProcess;
79         initGUI();
80     }
81
82     /**
83      * Method generated by Jigloo to initialize GUI components
84      */
85     private void initGUI() {
86         try {
87             this.setPreferredSize(new java.awt.Dimension(800, 500));
88             this.setLayout(null);
89             {
90                 lblItemExit = new JLabel();
91                 this.add(lblItemExit);
92                 lblItemExit.setText("4. Item Exit");
93                 lblItemExit.setFont(new java.awt.Font("Dialog",1,14));
94                 lblItemExit.setBounds(12, 12, 83, 16);
95             }
96             {
97                 lblItemExitVoucher = new JLabel();
98                 this.add(lblItemExitVoucher);
99                 lblItemExitVoucher.setText("Item Exit Voucher No.* :");
100                lblItemExitVoucher.setBounds(36, 58, 132, 16);
101            }
102            {
103                lblRequiredFields = new JLabel();
104                this.add(lblRequiredFields);
105                lblRequiredFields.setText("Fields marked with an * are required");
106                lblRequiredFields.setFont(new java.awt.Font("Dialog",0,10));
107                lblRequiredFields.setBounds(12, 28, 205, 16);
108            }
}
```

```
109     {
110         txtVoucher = new JTextField();
111         this.add(txtVoucher);
112         txtVoucher.setBounds(180, 54, 188, 20);
113         txtVoucher.setSize(206, 24);
114         txtVoucher.setDocument(new TextFieldDigitLimit(8));
115         txtVoucher.setMinimumSize(new java.awt.Dimension(4, 24));
116     }
117     {
118         lblDispatchedBy = new JLabel();
119         this.add(lblDispatchedBy);
120         lblDispatchedBy.setText("Dispatched By:");
121         lblDispatchedBy.setBounds(445, 56, 84, 16);
122     }
123     {
124         txtDispatchedBy = new JTextField();
125         this.add(txtDispatchedBy);
126         txtDispatchedBy.setBounds(541, 52, 206, 24);
127         txtDispatchedBy.setText(user.getUsername());
128         txtDispatchedBy.setEditable(false);
129     }
130     {
131         lblDate = new JLabel();
132         this.add(lblDate);
133         lblDate.setText("Date:");
134         lblDate.setBounds(500, 94, 29, 16);
135     }
136     {
137         txtDate = new JTextField();
138         this.add(txtDate);
139         txtDate.setBounds(541, 90, 206, 24);
140         txtDate.setEditable(false);
141         int day = date.get(Calendar.DAY_OF_MONTH);
142         int month = date.get(Calendar.MONTH);
143         int year = date.get(Calendar.YEAR);
144         txtDate.setText(day + "/" + month + "/" + year);
```

```
145
146
147     lblRequestedBy = new JLabel();
148     this.add(lblRequestedBy);
149     lblRequestedBy.setText("Requested By* :");
150     lblRequestedBy.setBounds(79, 94, 89, 16);
151 }
152 {
153     txtRequestedBy = new JTextField();
154     this.add(txtRequestedBy);
155     txtRequestedBy.setBounds(180, 90, 206, 24);
156 }
157 {
158     spExitList = new JSeparator();
159     this.add(spExitList);
160     spExitList.setBounds(12, 127, 776, 10);
161 }
162 {
163     scpQuantities = new JScrollPane();
164     this.add(scpQuantities);
165     scpQuantities.setBounds(36, 188, 711, 256);
166     scpQuantities.setSize(725, 256);
167 {
168     TransactionProcessingTableModel tblQuantitiesModel =
169         new TransactionProcessingTableModel(itemsToProcess);
170     tblQuantities = new JTable();
171     scpQuantities.setViewportView(tblQuantities);
172     tblQuantities.setModel(tblQuantitiesModel);
173     tblQuantities.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
174     tblQuantities.addMouseListener(new MouseAdapter() {
175         public void mouseClicked(MouseEvent e) {
176             if (e.getClickCount() == 2) {
177                 JTable tblResults = (JTable)e.getSource();
178                 int row = tblResults.getSelectedRow();
179                 TransactionProcessingTableModel model =
180                     (TransactionProcessingTableModel)tblQuantities.getModel();
```

```
181         Item item = (Item) model.getSearchResults().get(row);
182         ItemDescription description = new ItemDescription(item);
183         description.setVisible(true);
184     }
185 }
186 );
187
188 //Set Column "Item ID" width
189 int vColIndex = 0;
190 TableColumn colID = tblQuantities.getColumnModel().getColumn(vColIndex);
191 int width = 100;
192 colID.setPreferredWidth(width);
193
194 //Set Column "Item Name" width
195 TableColumn colName = tblQuantities.getColumnModel().getColumn(1);
196 colName.setPreferredWidth(308);
197
198 //Set Column "UM" width
199 TableColumn colUM = tblQuantities.getColumnModel().getColumn(2);
200 colUM.setPreferredWidth(50);
201
202 //Set Column "In Stock" width
203 TableColumn colStock = tblQuantities.getColumnModel().getColumn(3);
204 colStock.setPreferredWidth(100);
205
206 //Set Column "Quantity" width
207 TableColumn colQuantity = tblQuantities.getColumnModel().getColumn(4);
208 colQuantity.setPreferredWidth(150);
209
210 }
211 }
212 {
213     btnNext = new JButton();
214     this.add(btnNext);
215     btnNext.setText("Next Step");
216     btnNext.setBounds(672, 456, 89, 26);
```

```
217     btnNext.addActionListener(new ActionListener() {
218         public void actionPerformed (ActionEvent e){
219             performTransactionProcessing();
220         }
221     });
222 }
223 {
224     lblQuantities = new JLabel();
225     this.add(lblQuantities);
226     lblQuantities.setText("5. Quantities");
227     lblQuantities.setFont(new java.awt.Font("Dialog",1,14));
228     lblQuantities.setBounds(12, 143, 96, 16);
229 }
230 {
231     lblQuantitiesInfo = new JLabel();
232     this.add(lblQuantitiesInfo);
233     lblQuantitiesInfo.setText("Fill the 'Quantity' column for each item to be removed. Make sure "
234         + "that this amount is not larger than the current stock or less than or equal to 0");
235     lblQuantitiesInfo.setFont(new java.awt.Font("Dialog",0,10));
236     lblQuantitiesInfo.setBounds(12, 160, 749, 16);
237 }
238 } catch (Exception e) {
239     e.printStackTrace();
240 }
241 }
242 /**
243 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
244 */
245
246 public void performTransactionProcessing(){
247     if(isCompleteInput() && isCorrectQuantity()){
248         processAllTransactions();
249         JOptionPane.showMessageDialog(this, "Transactions successfully processed",
250             "Success", JOptionPane.INFORMATION_MESSAGE);
251         ChangeScreen.setOptimalRouteScreen(user, itemsToProcess);
252     } else if(!isCorrectQuantity() && !isCompleteInput()) {
```

```
253     JOptionPane.showMessageDialog(this, "Some fields are missing and the \n"
254         + "quantity entered is incorrect", "Error", JOptionPane.ERROR_MESSAGE);
255 } else if(!isCompleteInput()){
256     JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);
257 } else if(!isCorrectQuantity()){
258     JOptionPane.showMessageDialog(this, "One or more quantities entered are incorrect",
259         "Error", JOptionPane.ERROR_MESSAGE);
260 }
261 }
262 /**
263 * Processes all the transactions
264 */
265 public void processAllTransactions(){
266     TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
267     TransactionRecord[] transactionsToProcess = new TransactionRecord[model.getRowCount()];
268
269     for(int i=0;i<transactionsToProcess.length;i++){
270         transactionsToProcess[i] = new TransactionRecord();
271         transactionsToProcess[i].setType(ApplicationConstants.ITEM_EXIT);      //deals with item withdrawals
272         transactionsToProcess[i].setDocument(txtVoucher.getText());
273         transactionsToProcess[i]. setDate(this.date);
274         transactionsToProcess[i].setWarehouseWorker(txtDispatchedBy.getText());
275         transactionsToProcess[i].setThirdParty(txtRequestedBy.getText());
276         transactionsToProcess[i].setQuantity(((Integer)model.getValueAt(i, 4)).intValue());
277         transactionsToProcess[i].setItem((Item)model.getSearchResults().get(i));
278         transactionsToProcess[i].processTransaction();
279     }
280 }
281 }
282 /**
283 * Checks if the quantity entered is not greater than the quantity available
284 * @return true if the quantity is correct, else false
285 */
286 public boolean isCorrectQuantity(){
```

```
289 TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
290 for(int i=0;i<model.getRowCount();i++){
291     TransactionRecord record = model.getTransactions()[i];
292     Item item = (Item)model.getSearchResults().get(i);
293     if(record.getQuantity() > Transactions.getQuantityInStock(item) || record.getQuantity() <= 0){
294         return false;
295     }
296 }
297
298     return true;
299 }
300
301 /**
302 * Checks if all the necessary fields have been filled in
303 * @return true if all necessary fields have been filled in, else false
304 */
305 public boolean isCompleteInput(){
306     if(txtVoucher.getText().equals("") || txtRequestedBy.getText().equals("")){
307         return false;
308     } else {
309         return true;
310     }
311 }
312 }
313 }
314 }
```

```
1 import java.io.RandomAccessFile;
2 import java.util.ArrayList;
3 import java.util.GregorianCalendar;
4
5 /**
6 * -----
7 * Warehouse Application
8 * Class contains methods relevant to Item processing
9 * @author Alvaro Morales
10 * @date 10/07/2010
11 * @school Markham College
12 * @IDE Eclipse SDK
13 * @computer IBM ThinkPad R52
14 * -----
15 */
16
17 public class Items {
18
19     /**
20      * Gets an Item from the Items Random Access File given its position in the file
21      * @param position - the position of the item in the file (the Item's ID)
22      * @return the Item at that position, or null if found or not matching the ID provided
23      */
24     public static Item getItemFromFile(int position){
25         try{
26             RandomAccessFile file = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "r");
27
28             file.seek(position * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
29
30             int ID = file.readInt();
31
32             if(ID == position){           //The correct item has been found
33                 String code = removeExtraSpaces(file.readUTF());
34                 String name = removeExtraSpaces(file.readUTF());
35                 short groupCode = file.readShort();
36                 short umCode = file.readShort();
```

```
37     String description = removeExtraSpaces(file.readUTF());
38     byte warehouseNumber = file.readByte();
39     byte aisleNumber = file.readByte();
40     byte columnNumber = file.readByte();
41     char row = file.readChar();
42     String createdBy = removeExtraSpaces(file.readUTF());
43     short year = file.readShort();
44     byte month = file.readByte();
45     byte day = file.readByte();
46
47     Group group = getGroup(groupCode);
48     UM um = getUM(umCode);
49     Location location = new Location(warehouseNumber, aisleNumber, columnNumber, row);
50     GregorianCalendar creationDate = new GregorianCalendar(year, month, day);
51
52     return new Item(ID, code, name, group, um, description, location, createdBy, creationDate);
53 } else {
54     return null;
55 }
56
57 } catch (Exception e){
58     return null;
59 }
60
61 }
62
63 /**
64 * Sequentially searches the Items file to search for an item by group
65 * @param groupCode - the code of the group (search query)
66 * @return an ArrayList of items belonging to that group
67 */
68 public static ArrayList searchByGroup(short groupCode){
69     try{
70         ArrayList searchResults = new ArrayList();
71         RandomAccessFile file = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "r");
72     }
```

```
73     int position;
74     int numberOfRecords = (int)file.length() / ApplicationConstants.ITEMS_FILE_RECORD_LENGTH;
75
76     for(int i=0;i<numberOfRecords;i++){
77         file.seek(i * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
78         position = file.readInt();
79         if(position!=-999){           //if record is not empty
80             file.seek(position * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
81             Item item = getItemFromFile(position);
82             if(item.getGroup().getCode() == groupCode){
83                 searchResults.add(item);
84             }
85         }
86     }
87     searchResults.trimToSize();
88     return searchResults;
89 } catch (Exception e){
90     return null;
91 }
92 }
93
94 /**
95 * Sequentially searches the Items file to search for an item by location
96 * @param location - the location of the item (search query)
97 * @return an ArrayList of items stored at that location
98 */
99 public static ArrayList searchByLocation(Location location){
100    try{
101        ArrayList searchResults = new ArrayList();
102        RandomAccessFile file = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "r");
103
104        int position;
105        int numberOfRecords = (int)file.length() / ApplicationConstants.ITEMS_FILE_RECORD_LENGTH;
106
107        for(int i=0;i<numberOfRecords;i++){
108            file.seek(i * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
```

```
109     position = file.readInt();
110     if(position!=-999){           //if record is not empty
111         file.seek(position * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);
112         Item item = getItemFromFile(position);
113         if(item.getLocation().getAisle() == location.getAisle() &&
114             item.getLocation().getRow() == location.getRow() &&
115             item.getLocation().getColumn() == location.getColumn())){
116             searchResults.add(item);
117         }
118     }
119 }
120
121     searchResults.trimToSize();
122     return searchResults;
123 } catch (Exception e){
124     return null;
125 }
126 }
127
128 /**
129 * Gets a Group object from the Groups array given its Group code
130 * @param groupCode - the code of the Group
131 * @return the Group matching the code provided, or null if not found
132 */
133 public static Group getGroup(short groupCode){
134     for (int i = 0; i < MainScreen.groups.length; i++){
135         if(MainScreen.groups[i].getCode() == groupCode){
136             return MainScreen.groups[i];
137         }
138     }
139 }
140
141     return null;
142 }
143
144 /**
```

```
145     * Gets a UM object from the UM array given its UM code
146     * @param umCode - the code of the Unit of Measurement (UM)
147     * @return the UM matching the code provided, or null if not found
148     */
149    public static UM getUM(short umCode){
150        for (int i = 0; i < MainScreen.umArray.length; i++){
151            if(MainScreen.umArray[i].getCode() == umCode){
152                return MainScreen.umArray[i];
153            }
154        }
155
156        return null;
157    }
158
159    /**
160     * Removes the extra spaces at the end of a String
161     * @param s - the String to remove extra spaces from
162     * @return the same String without the extra spaces at the end
163     */
164    public static String removeExtraSpaces(String s){
165        while(s.charAt(s.length() -1) == ' '){
166            s = s.substring(0, s.length() -1);
167        }
168        return s;
169    }
170
171 }
```

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.table.*;
5 import java.util.*;
6
7 /**
8 * -----
9 * Warehouse Application
10 * The Item Search screen to search for an item
11 * @author Alvaro Morales
12 * @date 28/06/2010
13 * @school Markham College
14 * @IDE Eclipse SDK
15 * @computer IBM ThinkPad R52
16 * -----
17 */
18 public class ItemSearch extends javax.swing.JPanel {
19
20     /**
21      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
22      */
23
24     {
25         //Set Look & Feel
26         try {
27             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
28         } catch(Exception e) {
29             e.printStackTrace();
30         }
31     }
32
33     /**
34      * GUI components generated by Jigloo
35      */
36 }
```

```
37     private JLabel lblSearch;
38     private JLabel lblSearchBy;
39     private JComboBox cmbSearchCriteria;
40     private JButton btnSearch;
41     private JLabel lblName;
42     private JLabel lblCode;
43     private JComboBox cmbGroup;
44     private JLabel lblGroup;
45     private JLabel lblCodeSearchInfo;
46     private JTextField txtSearchByName;
47     private JRadioButton rbtnPartialMatch;
48     private JRadioButton rbtnExactMatch;
49     private ButtonGroup btnGroupSearchType;
50     private JLabel lblColumn;
51     private JLabel lblRow;
52     private JLabel lblAisle;
53     private JComboBox cmbColumn;
54     private JComboBox cmbRow;
55     private JComboBox cmbAisle;
56     private JLabel lblLocation;
57     private JTable tblSearchResults;
58     private JScrollPane scpResultsTableHolder;
59     private JLabel lblSearchHelp;
60     private JLabel lblSearchResults;
61     private JSeparator spSearchFields;
62     private JTextField txtSearchByCode;
63
64     /**
65      * The user that logged in
66      */
67     private User user;
68
69     /**
70      * Constructs a new ItemSearch object
71      * @param user - the user that logged in
72      */
```

```
73     public ItemSearch(User user) {
74         super();
75         this.user = user;
76         initGUI();
77     }
78
79     /**
80      * Method generated by Jigloo to initialize GUI components
81      */
82     private void initGUI() {
83         try {
84             setPreferredSize(new Dimension(400, 300));
85             this.setSize(800, 500);
86             this.setLayout(null);
87             {
88                 lblSearch = new JLabel();
89                 this.add(lblSearch);
90                 lblSearch.setText("Item Enquiry");
91                 lblSearch.setBounds(12, 12, 164, 21);
92                 lblSearch.setFont(new java.awt.Font("Dialog", 1, 14));
93             }
94
95             {
96                 ComboBoxModel cmbSearchCriteriaModel =
97                     new DefaultComboBoxModel(
98                         new String[] { "Code", "Name", "Group", "Location" });
99                 cmbSearchCriteria = new JComboBox();
100                this.add(cmbSearchCriteria);
101                cmbSearchCriteria.setModel(cmbSearchCriteriaModel);
102                cmbSearchCriteria.setBounds(119, 44, 117, 25);
103                cmbSearchCriteria.addActionListener(new ActionListener() {
104                    public void actionPerformed (ActionEvent e){
105                        changeSearchCriteria();
106                    }
107                });
108            }
109        }
```

```
109
110     txtSearchByCode = new JTextField();
111     this.add(txtSearchByCode);
112     txtSearchByCode.setBounds(119, 81, 266, 20);
113     txtSearchByCode.setSize(266, 24);
114     txtSearchByCode.setDocument(new TextFieldDigitLimit(6));
115     txtSearchByCode.setVisible(true);
116 }
117 {
118     lblSearchBy = new JLabel();
119     this.add(lblSearchBy);
120     lblSearchBy.setText("Search By:");
121     lblSearchBy.setBounds(46, 48, 61, 16);
122 }
123 {
124     spSearchFields = new JSeparator();
125     this.add(spSearchFields);
126     spSearchFields.setBounds(7, 174, 776, 10);
127 }
128 {
129     btnSearch = new JButton();
130     this.add(btnSearch);
131     btnSearch.setText("Search");
132     btnSearch.setBounds(119, 130, 102, 26);
133     btnSearch.addActionListener(new ActionListener() {
134         public void actionPerformed (ActionEvent e){
135             performSearch();
136         }
137     });
138 }
139 {
140     lblSearchResults = new JLabel();
141     this.add(lblSearchResults);
142     lblSearchResults.setText("Search Results");
143     lblSearchResults.setFont(new java.awt.Font("Dialog",1,14));
144     lblSearchResults.setBounds(12, 190, 119, 16);
```

```
145 }
146 {
147     lblSearchHelp = new JLabel();
148     this.add(lblSearchHelp);
149     lblSearchHelp.setText("Double click on an item to view its information");
150     lblSearchHelp.setFont(new java.awt.Font("Dialog",0,10));
151     lblSearchHelp.setBounds(12, 212, 259, 16);
152 }
153 {
154     scpResultsTableHolder = new JScrollPane();
155     this.add(scpResultsTableHolder);
156     scpResultsTableHolder.setBounds(46, 240, 712, 215);
157 {
158     TableModel tblSearchResultsModel = new SearchResultsTableModel(null);
159     tblSearchResults = new JTable();
160     scpResultsTableHolder.setViewportView(tblSearchResults);
161     tblSearchResults.setModel(tblSearchResultsModel);
162     tblSearchResults.setColumnSelectionAllowed(false);
163     tblSearchResults.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
164     tblSearchResults.addMouseListener(new MouseAdapter() {
165         public void mouseClicked(MouseEvent e) {
166             if (e.getClickCount() == 2) {
167                 JTable tblResults = (JTable)e.getSource();
168                 int row = tblResults.getSelectedRow();
169                 SearchResultsTableModel model =
170                     (SearchResultsTableModel)tblSearchResults.getModel();
171                 Item item = (Item)model.getSearchResults().get(row);
172                 ItemDescription description = new ItemDescription(item, user);
173                 description.setVisible(true);
174             }
175         }
176     });
177 }
178 {
179     lblLocation = new JLabel();
180     this.add(lblLocation);
```

```
181     lblLocation.setText("Location:");
182     lblLocation.setBounds(54, 85, 54, 16);
183     lblLocation.setVisible(false);
184 }
185 {
186     ComboBoxModel cmbAisleModel = new DefaultComboBoxModel(MainScreen.locations.getAisles());
187     cmbAisle = new JComboBox();
188     this.add(cmbAisle);
189     cmbAisle.setModel(cmbAisleModel);
190     cmbAisle.setBounds(119, 80, 80, 26);
191     cmbAisle.setVisible(false);
192 }
193 {
194     ComboBoxModel cmbColumnModel = new DefaultComboBoxModel(MainScreen.locations.getRows());
195     cmbRow = new JComboBox();
196     this.add(cmbRow);
197     cmbRow.setModel(cmbColumnModel);
198     cmbRow.setBounds(210, 81, 80, 25);
199     cmbRow.setVisible(false);
200 }
201 {
202     ComboBoxModel cmbColumnModel = new DefaultComboBoxModel(MainScreen.locations.getColumns());
203     cmbColumn = new JComboBox();
204     this.add(cmbColumn);
205     cmbColumn.setModel(cmbColumnModel);
206     cmbColumn.setBounds(303, 81, 82, 25);
207     cmbColumn.setVisible(false);
208 }
209 {
210     lblAisle = new JLabel();
211     this.add(lblAisle);
212     lblAisle.setText("Aisle");
213     lblAisle.setBounds(144, 105, 24, 16);
214     lblAisle.setFont(new java.awt.Font("Dialog", 0, 10));
215     lblAisle.setVisible(false);
216 }
```

```
217     {
218         lblRow = new JLabel();
219         this.add(lblRow);
220         lblRow.setText("Row");
221         lblRow.setBounds(244, 105, 25, 16);
222         lblRow.setFont(new java.awt.Font("Dialog",0,10));
223         lblRow.setVisible(false);
224     }
225     {
226         lblColumn = new JLabel();
227         this.add(lblColumn);
228         lblColumn.setText("Column");
229         lblColumn.setBounds(325, 105, 40, 16);
230         lblColumn.setFont(new java.awt.Font("Dialog",0,10));
231         lblColumn.setVisible(false);
232     }
233     {
234         rbtnExactMatch = new JRadioButton();
235         rbtnExactMatch.setText("Exact Match");
236         rbtnExactMatch.setBounds(509, 81, 95, 24);
237         rbtnExactMatch.setVisible(false);
238     }
239     {
240         rbtnPartialMatch = new JRadioButton();
241         rbtnPartialMatch.setText("Partial Match");
242         rbtnPartialMatch.setBounds(394, 81, 100, 24);
243         rbtnPartialMatch.setSelected(true);
244         rbtnPartialMatch.setVisible(false);
245     }
246     {
247         ButtonGroup btnGroupSearchType = new ButtonGroup();
248         btnGroupSearchType.add(rbtnExactMatch);
249         btnGroupSearchType.add(rbtnPartialMatch);
250     }
251     {
252         txtSearchByName = new JTextField();
```

```
253     this.add(txtSearchByName);
254     txtSearchByName.setVisible(false);
255     txtSearchByName.setBounds(119, 81, 266, 20);
256     txtSearchByName.setSize(266, 24);
257     txtSearchByName.setVisible(false);
258 }
259 {
260     this.add(rbtnExactMatch);
261     this.add(rbtnPartialMatch);
262 }
263 {
264     lblCodeSearchInfo = new JLabel();
265     this.add(lblCodeSearchInfo);
266     lblCodeSearchInfo.setText("Only digits");
267     lblCodeSearchInfo.setBounds(392, 88, 60, 13);
268     lblCodeSearchInfo.setFont(new java.awt.Font("Dialog", 0, 10));
269 }
270 {
271     lblGroup = new JLabel();
272     this.add(lblGroup);
273     lblGroup.setText("Group:");
274     lblGroup.setBounds(70, 85, 37, 16);
275     lblGroup.setVisible(false);
276 }
277 {
278     ComboBoxModel cmbGroupModel = new DefaultComboBoxModel(MainScreen.groups);
279     cmbGroup = new JComboBox();
280     this.add(cmbGroup);
281     cmbGroup.setModel(cmbGroupModel);
282     cmbGroup.setBounds(119, 81, 172, 25);
283     cmbGroup.setVisible(false);
284 }
285 {
286     lblCode = new JLabel();
287     this.add(lblCode);
288     lblCode.setText("Code:");
```

```
289         lblCode.setBounds(75, 85, 32, 16);
290     }
291     {
292         lblName = new JLabel();
293         this.add(lblName);
294         lblName.setText("Name:");
295         lblName.setBounds(71, 85, 36, 16);
296         lblName.setVisible(false);
297     }
298 }
299 }
300 } catch (Exception e) {
301     e.printStackTrace();
302 }
303 }
304
305
306 /**
307 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
308 */
310
311 /**
312 * Updates GUI elements to match the search criteria and type
313 */
314 public void changeSearchCriteria(){
315     String searchCriteria = (String)cmbSearchCriteria.getSelectedItem();
316
317     if (searchCriteria.equals("Code")) {
318         //Enable and display GUI elements for a search by code
319         lblCode.setVisible(true);
320         txtSearchByCode.setVisible(true);
321         txtSearchByCode.setDocument(new TextFieldDigitLimit(6));
322         lblCodeSearchInfo.setVisible(true);
323
324         //Disable and make invisible GUI elements for other types of searches
```

```
325     lblName.setVisible(false);
326     txtSearchByName.setVisible(false);
327     rbtnPartialMatch.setVisible(false);
328     rbtnExactMatch.setVisible(false);
329
330     lblGroup.setVisible(false);
331     cmbGroup.setVisible(false);
332
333     lblLocation.setVisible(false);
334     lblAisle.setVisible(false);
335     lblRow.setVisible(false);
336     lblColumn.setVisible(false);
337     cmbAisle.setVisible(false);
338     cmbRow.setVisible(false);
339     cmbColumn.setVisible(false);
340
341     this.validate();
342 } else if (searchCriteria.equals("Name")){
343     //Enable and display GUI elements for a search by name
344     lblName.setVisible(true);
345     txtSearchByName.setVisible(true);
346     txtSearchByName.setDocument(new TextFieldLimit(200));
347     rbtnPartialMatch.setVisible(true);
348     rbtnExactMatch.setVisible(true);
349
350     //Disable and make invisible GUI elements for other types of searches
351     lblCode.setVisible(false);
352     txtSearchByCode.setVisible(false);
353     lblCodeSearchInfo.setVisible(false);
354
355     lblGroup.setVisible(false);
356     cmbGroup.setVisible(false);
357
358     lblLocation.setVisible(false);
359     lblAisle.setVisible(false);
360     lblRow.setVisible(false);
```

```
361     lblColumn.setVisible(false);
362     cmbAisle.setVisible(false);
363     cmbRow.setVisible(false);
364     cmbColumn.setVisible(false);
365
366     this.validate();
367 } else if (searchCriteria.equals("Group")){
368     //Enable and display GUI elements for a search by group
369     lblGroup.setVisible(true);
370     cmbGroup.setVisible(true);
371
372     //Disable and make invisible GUI elements for other types of searches
373     lblCode.setVisible(false);
374     txtSearchByCode.setVisible(false);
375     lblCodeSearchInfo.setVisible(false);
376
377     lblName.setVisible(false);
378     txtSearchByName.setVisible(false);
379     rbtnPartialMatch.setVisible(false);
380     rbtnExactMatch.setVisible(false);
381
382     lblLocation.setVisible(false);
383     lblAisle.setVisible(false);
384     lblRow.setVisible(false);
385     lblColumn.setVisible(false);
386     cmbAisle.setVisible(false);
387     cmbRow.setVisible(false);
388     cmbColumn.setVisible(false);
389
390     this.validate();
391 } else if(searchCriteria.equals("Location")){
392     //Enable and display GUI elements for a search by location
393     lblLocation.setVisible(true);
394     lblAisle.setVisible(true);
395     lblRow.setVisible(true);
396     lblColumn.setVisible(true);
```

```
397     cmbAisle.setVisible(true);
398     cmbRow.setVisible(true);
399     cmbColumn.setVisible(true);
400
401     //Disable and make invisible GUI elements for other types of searches
402     lblCode.setVisible(false);
403     txtSearchByCode.setVisible(false);
404     lblCodeSearchInfo.setVisible(false);
405
406     lblName.setVisible(false);
407     txtSearchByName.setVisible(false);
408     rbtnPartialMatch.setVisible(false);
409     rbtnExactMatch.setVisible(false);
410
411     lblGroup.setVisible(false);
412     cmbGroup.setVisible(false);
413
414     this.validate();
415 }
416 }
417
418 /**
419 * Performs a search by code on the Code Index Binary Tree.
420 * Updates the table to show the items matching.
421 * If there is no match, the user is alerted.
422 * @param codeToFind - the code of the item to find (the search query)
423 */
424 public void performSearchByCode(String codeToFind) {
425
426     Item item = MainScreen.codeIndexTree.search(codeToFind);
427
428     if(item == null){
429         displayNoResultsMessage();
430         txtSearchByCode.setText("");
431     } else {
432         ArrayList list = new ArrayList();
```

```
433     list.add(item);
434     list.trimToSize();
435     tblSearchResults.setModel(new SearchResultsTableModel(list));
436     tblSearchResults.validate();
437 }
438 }
439 /**
440 * Performs a search by exact name on the Name Index Binary Tree.
441 * Updates the table to show the items matching.
442 * If there is no match, the user is alerted.
443 * @param exactName - the exact name of the item to find (the search query)
444 */
445 public void performExactSearchByName(String exactName) {
446     Item item = MainScreen.nameIndexTree.search(exactName);
447
448     if(item == null){
449         displayNoResultsMessage();
450         txtSearchByName.setText("");
451     } else {
452         ArrayList list = new ArrayList();
453         list.add(item);
454         list.trimToSize();
455         tblSearchResults.setModel(new SearchResultsTableModel(list));
456         tblSearchResults.validate();
457     }
458 }
459 }
460 /**
461 * Performs a search by partial name on the Name Index Binary Tree.
462 * @param partialName - the partial match of the name of the item to find (the search query)
463 * Updates the table to show the items matching.
464 * If there is no match, the user is alerted.
465 */
466 public void performPartialSearchByName(String partialName) {
```

```
469     ArrayList searchResults = MainScreen.nameIndexTree.partialSearch(partialName);
470
471     if(searchResults == null){
472         displayNoResultsMessage();
473         txtSearchByName.setText("");
474     } else {
475         tblSearchResults.setModel(new SearchResultsTableModel(searchResults));
476         tblSearchResults.validate();
477     }
478 }
479
480 /**
481 * Alerts the user that no results matching the query have been found and clears the table
482 */
483 public void displayNoResultsMessage(){
484     JOptionPane.showMessageDialog(this, "No results", "Error", JOptionPane.ERROR_MESSAGE);
485     tblSearchResults.setModel(new SearchResultsTableModel(null));
486     tblSearchResults.validate();
487 }
488
489 /**
490 * Performs a search depending on the criteria selected
491 */
492 public void performSearch(){
493     if (txtSearchByCode.isVisible()) {
494         if(!txtSearchByCode.getText().equals("")){
495             if(txtSearchByCode.getText().length() != 6){
496                 JOptionPane.showMessageDialog(this, "You must enter a 6 digit code query",
497                     "Error", JOptionPane.ERROR_MESSAGE);
498             } else {
499                 performSearchByCode(txtSearchByCode.getText());
500             }
501         } else {
502             JOptionPane.showMessageDialog(this, "You must enter a search query",
503                 "Error", JOptionPane.ERROR_MESSAGE);
504         }
505     }
506 }
```

```
505     } else if (txtSearchByName.isVisible()) {
506         if (!txtSearchByName.getText().equals("")) {
507             if (rbtnExactMatch.isSelected()) {
508                 performExactSearchByName(txtSearchByName.getText());
509             } else {
510                 performPartialSearchByName(txtSearchByName.getText());
511             }
512         } else {
513             JOptionPane.showMessageDialog(this, "You must enter a search query",
514                                         "Error", JOptionPane.ERROR_MESSAGE);
515         }
516     } else if (cmbGroup.isVisible()) {
517         Group group = (Group)cmbGroup.getSelectedItem();
518         performSearchByGroup(group.getCode());
519     } else if (cmbAisle.isVisible()) {
520         performSearchByLocation();
521     }
522 }
523 /**
524 * Opens an the description of a selected item (row in a table)
525 * @param row - the selected row index
526 */
527 public void openItemDescription(int row){
528     SearchResultsTableModel model = (SearchResultsTableModel)tblSearchResults.getModel();
529     Item item = (Item)model.getSearchResults().get(row);
530     ItemDescription description = new ItemDescription(item, user);
531     description.setVisible(true);
532 }
533 /**
534 * Performs a search by group, updating the table with the search results
535 * @param groupCode - the code of the group (search query)
536 */
537 public void performSearchByGroup(short groupCode){
538     ArrayList searchResults = Items.searchByGroup(groupCode);
```

```
541     if(searchResults != null && !searchResults.isEmpty()){
542         tblSearchResults.setModel(new SearchResultsTableModel(searchResults));
543         tblSearchResults.validate();
544     } else {
545         displayNoResultsMessage();
546     }
547 }
548
549
550 /**
551 * Performs a search by group, updating the table with the search results
552 * @param groupCode - the code of the group (search query)
553 */
554 public void performSearchByLocation(){
555     byte warehouse = 1;
556     byte aisle = ((Integer)cmbAisle.getSelectedItem()).byteValue();
557     char row = ((String)cmbRow.getSelectedItem()).charAt(0);
558     byte column = ((Integer)cmbColumn.getSelectedItem()).byteValue();
559
560     Location location = new Location(warehouse, aisle, column, row);
561     ArrayList searchResults = Items.searchByLocation(location);
562     if(!searchResults.isEmpty() || searchResults == null){
563         tblSearchResults.setModel(new SearchResultsTableModel(searchResults));
564         tblSearchResults.validate();
565     } else {
566         displayNoResultsMessage();
567     }
568 }
569
570 }
```

```
1 import java.io.*;
2
3 /**
4 * -----
5 * Warehouse Application
6 * Filters the files in a JFileChooser to only display .jpg images
7 * @author Alvaro Morales
8 * with references to "How to Use File Choosers" from the Java Tutorial found in
9 * http://java.sun.com/docs/books/tutorial/uiswing/components/filechooser.html
10 * @date 02/07/2010
11 * @school Markham College
12 * @IDE Eclipse SDK
13 * @computer IBM ThinkPad R52
14 * -----
15 */
16
17 /**
18 * Filters the files in a JFileChooser to only display .jpg images
19 */
20 public class JpgFilter extends javax.swing.filechooser.FileFilter{
21
22     /**
23      * Constructs a new JpgFilter object
24      */
25     public JpgFilter(){
26         super();
27     }
28
29     /**
30      * Filters a file (accepts it only if its extension is a .jpg image or it is a directory)
31      * @param file - the file to filter
32      * @return true if the file has the extension .jpg or is a directory, else false
33      */
34     public boolean accept(File file){
35         if(file.isDirectory()){
36             return true;
```

```
37     }
38
39     String extension = getExtension(file);
40     if(extension!= null){
41         if(extension.equals("jpg")){
42             return true;
43         } else {
44             return false;
45         }
46     } else {
47         return false;
48     }
49
50 }
51
52 /**
53 * Gets the extension of a file
54 * @param file - the file that will be processed to get its extension
55 * @return the extension of a file
56 */
57 public String getExtension(File file){
58     String fileName = file.getName();
59     String extension = null;
60     int extensionStart = fileName.lastIndexOf('.'); //Gets starting position of the '.extension' in the file
61
62     if(extensionStart > 0 && extensionStart < fileName.length()-1){ //Checks for appropriate length
63         extension = fileName.substring(extensionStart+1).toLowerCase();
64     }
65
66     return extension;
67 }
68
69 /**
70 * Gets the description of this FileFilter
71 * @return the description of this FileFilter
72 }
```

```
73     */
74     public String getDescription() {
75         return "JPG Images";
76     }
77
78 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the Location object that has information regarding an item's location in the warehouse
5  * @author Alvaro Morales
6  * @date 24/06/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12
13 public class Location implements Comparable {
14
15     /**
16      * The warehouse number where the item is stored
17      */
18     private byte warehouse;
19
20     /**
21      * The aisle where the item is stored
22      */
23     private byte aisle;
24
25     /**
26      * The column of the aisle where the item is stored
27      */
28     private byte column;
29
30     /**
31      * The row of the aisle where the item is stored
32      */
33     private char row;
34
35
36     /**
```

```
37     * Constructs a new Location object
38     * @param aisle - the aisle where the item is stored
39     * @param column - the column of the aisle where the item is stored
40     * @param row - the row of the aisle where the item is stored
41     * @param position - the position in a row/column location where the item is stored
42     */
43 public Location(byte aisle, byte column, char row){
44     this.warehouse = 1;          //this application deals with item stored in Warehouse 1
45     this.aisle = aisle;
46     this.column = column;
47     this.row = row;
48 }
49
50 /**
51     * Constructs a new Location object
52     * @param aisle - the aisle where the item is stored
53     * @param column - the column of the aisle where the item is stored
54     * @param row - the row of the aisle where the item is stored
55     * @param position - the position in a row/column location where the item is stored
56     */
57 public Location(byte warehouse, byte aisle, byte column, char row){
58     this.warehouse = warehouse;
59     this.aisle = aisle;
60     this.column = column;
61     this.row = row;
62 }
63
64
65 /**
66     * Gets the aisle number where the item is stored
67     * @return the aisle number where the item is stored
68     */
69 public byte getAisle() {
70     return aisle;
71 }
72 }
```

```
73  /**
74   * Sets the aisle number where the item is stored
75   * @param aisle - the aisle number where the item is stored, of type byte
76   */
77  public void setAisle(byte aisle) {
78      this.aisle = aisle;
79  }
80
81 /**
82  * Gets the column where the item is stored
83  * @return the column where the item is stored
84  */
85  public byte getColumn() {
86      return column;
87  }
88
89 /**
90  * Sets the column where the item is stored
91  * @param column - the column where the item is stored, of type byte
92  */
93  public void setColumn(byte column) {
94      this.column = column;
95  }
96
97 /**
98  * Gets the row where the item is stored
99  * @return the row where the item is stored
100 */
101 public char getRow() {
102     return row;
103 }
104
105 /**
106  * Sets the row where the item is stored
107  * @param row - the row where the item is stored, of type char
108  */
```

```
109     public void setRow(char row) {
110         this.row = row;
111     }
112
113     /**
114      * Gets the warehouse number where the item is stored
115      * @return the warehouse number where the item is stored
116      */
117     public byte getWarehouse() {
118         return warehouse;
119     }
120
121     /**
122      * Sets the warehouse number where the item is stored
123      * @param warehouse - the warehouse number where the item is stored
124      */
125     public void setWarehouse(byte warehouse) {
126         this.warehouse = warehouse;
127     }
128
129     /**
130      * Outputs the Location object in the format A01.AA.RCC
131      * where AA is the aisle number, R is the row char and CC is the column number
132      * @return the Location of the item
133      */
134     public String toString(){
135
136         String aisle = "" + getAisle();
137         if(aisle.length()!= 2){
138             aisle = "0" + aisle;
139         }
140
141         String column = "" + getColumn();
142         if(column.length()!= 2){
143             column = "0" + column;
144         }
145     }
```

```
145     return ("A01." + aisle + "." + getRow() + column);
146 }
147 }
148 /**
149 * Checks if this Location object is equal to another one
150 * @param o - the object to compare
151 * @return true if the aisle, row and column number match, else false
152 */
153 public boolean equals(Object o){
154     if(!(o instanceof Location)){
155         return false;
156     }
157     Location location = (Location)o;
158     return (location.getAisle() == getAisle() && location.getRow() == getRow()
159             && location.column == getColumn());
160 }
161 /**
162 * Compares this Location to another Location
163 * @param o - the object (of type Location) to compare to
164 * @return 0 if equal, 1 if this object is greater than o, -1 if this object is less than o
165 */
166 public int compareTo(Object o) throws ClassCastException{
167     if (o instanceof Location) {
168         Location location = (Location) o;
169         if>equals(o)){
170             return 0;
171         } else if(getAisle() < location.getAisle()){
172             return -1;
173         } else if(getAisle() > location.getAisle()){
174             return 1;
175         } else if(getColumn() < location.getColumn()){
176             return -1;
177         } else if (getColumn() > location.getColumn()){
178             return 1;
179         }
180     }
```

```
181         return 1;
182     } else if (getRow() < location.getRow()) {
183         return -1;
184     } else {
185         return 1;
186     }
187 } else {
188     throw new ClassCastException();
189 }
190 }
191
192
193 }
```

```
1 import javax.swing.*;
2 import java.io.*;
3 import java.util.*;
4 import java.awt.event.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * Constructs the Login Screen and handles user login
10 * @author Alvaro Morales
11 * @date 04/06/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class LoginScreen extends javax.swing.JFrame {
18
19 /**
20 * GUI components generated by Jigloo
21 */
22
23 private JLabel lblWarehouse;
24 private JSeparator spLogin;
25 private JButton btnLogin;
26 private JPasswordField txtPassword;
27 private JTextField txtUsername;
28 private JLabel lblPassword;
29 private JLabel lblUsername;
30
31 /**
32 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
33 */
34
35 {
36     //Set Look & Feel
```

```
37     try {
38         javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
39     } catch(Exception e) {
40         e.printStackTrace();
41     }
42 }
43
44 /**
45 * Auto-generated main method to display this JFrame
46 */
47 public static void main(String[] args) {
48     SwingUtilities.invokeLater(new Runnable() {
49         public void run() {
50             LoginScreen inst = new LoginScreen();
51             inst.setLocationRelativeTo(null);
52             inst.setVisible(true);
53         }
54     });
55 }
56
57 /**
58 * Constructs a new LoginScreen object
59 */
60 public LoginScreen() {
61     super();
62     initGUI();
63 }
64
65 /**
66 * Method generated by Jigloo to initialize GUI components
67 */
68 private void initGUI() {
69     try {
70         this.setFocusCycleRoot(false);
71         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
72         this.setTitle("Warehouse Application");
```

```
73     this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
74         ApplicationConstants.WAREHOUSE_LOGO)).getImage());
75     getContentPane().setLayout(null);
76     this.setResizable(false);
77     {
78         lblWarehouse = new JLabel();
79         getContentPane().add(lblWarehouse);
80         lblWarehouse.setBounds(25, 24, 250, 181);
81         lblWarehouse.setIcon(new ImageIcon(getClass().getClassLoader().getResource(
82             ApplicationConstants.IMAGES_FOLDER + "WarehousePicture.jpg")));
83     }
84     {
85         lblUsername = new JLabel();
86         getContentPane().add(lblUsername);
87         lblUsername.setText("Username:");
88         lblUsername.setBounds(25, 244, 62, 16);
89     }
90     {
91         lblPassword = new JLabel();
92         getContentPane().add(lblPassword);
93         lblPassword.setText("Password:");
94         lblPassword.setBounds(25, 278, 61, 16);
95     }
96     {
97         txtUsername = new JTextField();
98         getContentPane().add(txtUsername);
99         txtUsername.setBounds(105, 242, 165, 20);
100        txtUsername.setDocument(new TextFieldLimit(20));      //Maximum 20 characters
101        txtUsername.addActionListener(new ActionListener() {
102            public void actionPerformed (ActionEvent e){
103                performLogin(txtUsername.getText(), new String(txtPassword.getPassword()));
104            }
105        });
106    }
107    {
108        txtPassword = new JPasswordField();
```

```
109         getContentPane().add(txtPassword);
110         txtPassword.setBounds(104, 276, 166, 20);
111         txtPassword.setDocument(new TextFieldLimit(30));      //Maximum 30 characters
112         txtPassword.addActionListener(new ActionListener() {
113             public void actionPerformed (ActionEvent e){
114                 performLogin(txtUsername.getText(), new String(txtPassword.getPassword()));
115             }
116         });
117     }
118     {
119         btnLogin = new JButton();
120         getContentPane().add(btnLogin);
121         btnLogin.setText("Login");
122         btnLogin.setBounds(123, 308, 65, 26);
123         btnLogin.addActionListener(new ActionListener() {
124             public void actionPerformed (ActionEvent e){
125                 performLogin(txtUsername.getText(), new String(txtPassword.getPassword()));
126             }
127         });
128     }
129 }
130 {
131     spLogin = new JSeparator();
132     getContentPane().add(spLogin);
133     spLogin.setBounds(12, 226, 274, 10);
134 }
135 pack();
136 this.setSize(306, 373);
137 } catch (Exception e) {
138     e.printStackTrace();
139 }
140 }
141
142
143 /**
144 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
```

```
145     */
146
147
148     /**
149      * Method performed when the Login button is pressed
150      * Performs login by calling login(), and informs the user if login has been successful
151      * @param username - text input to the username JTextField
152      * @param password - text input to the password JPasswordField
153      */
154     private void performLogin(String username, String password){
155         User user = login(username, password);
156         if(user == null){
157             JOptionPane.showMessageDialog(this, "Invalid username or password.",
158                 "Login Error", JOptionPane.ERROR_MESSAGE);
159             txtPassword.setText("");
160             txtUsername.setText("");
161             txtUsername.grabFocus();
162         } else {
163             MainScreen main = new MainScreen(user);
164             main.setVisible(true);
165             this.dispose();
166         }
167     }
168
169     /**
170      * Compares user input for a username and password to registered users for the Login process
171      * @param username - the text input from the username JTextField
172      * @param password - the text input from the password JPasswordField
173      * @return if successful, the User that logged in. Else returns nothing
174      * @mastery achieves HL mastery factor 3 by searching the Users file
175      * to see if the username input by the user matches an existing username
176      */
177     public User login(String username, String password){
178
179         try{
180             File file = new File(ApplicationConstants.USERS_FILE);
```

```
181     FileReader reader = new FileReader(file);
182     BufferedReader buff = new BufferedReader(reader);
183
184     boolean eof = false;           //stores whether the end of the file has been reached
185
186     while(!eof){
187         String line = buff.readLine();
188         if (line == null){
189             eof = true;           //the end of the file has been reached
190         } else {
191             User user = readUser(line);
192
193             if(user.getUsername().equals(username) && user.getPassword().equals(password)
194                 && user.isEnabled() == true){
195                 return user;
196             } else if (user.getUsername().compareTo(username)>0){    //if less than, it has not been found
197                 return null;
198             }
199         }
200     }
201
202     buff.close();
203     reader.close();
204
205     return null;
206
207 } catch(Exception e){
208     return null;
209 }
210
211 }
212
213 /**
214 * Creates a User object from a tokenized String input
215 * @param line - a line read from the Users file
216 * @return - a User object
```

```
217     */
218     public User readUser(String line){
219         StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
220
221         if (tokenizer.countTokens() == 7){      //preliminary error checking
222             String username = tokenizer.nextToken();
223             String password = tokenizer.nextToken();
224             String fullName = tokenizer.nextToken();
225             boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
226             boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
227             boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
228             boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
229
230             return new User(username, password, fullName, admin, entry, exit, userEnabled);
231         }
232
233         return null;
234     }
235
236
237 /**
238 * Converts an int to a boolean
239 * @param 0 if false, 1 if true
240 * @return true if param is 1, else false
241 */
242 public boolean intToBoolean(int i){
243     return (i==1);
244 }
245
246 }
```

```
1 import java.io.*;
2 import java.util.StringTokenizer;
3 import javax.swing.*;
4 import java.util.Arrays;
5 import java.awt.*;
6
7 /**
8 * -----
9 * Warehouse Application
10 * Constructs the Main Screen of the application
11 * @author Alvaro Morales
12 * @date 04/06/2010
13 * @school Markham College
14 * @IDE Eclipse SDK
15 * @computer IBM ThinkPad R52
16 * -----
17 */
18 public class MainScreen extends javax.swing.JFrame {
19
20     /**
21      * Utilities that will be accessed throughout the application
22      */
23
24     /**
25      * The Binary Tree of Item indexes by name
26      */
27     public static IndexBTree nameIndexTree;
28
29     /**
30      * The Binary Tree of Item indexes by code
31      */
32     public static IndexBTree codeIndexTree;
33
34     /**
35      * The array of Item groups
36      */
```

```
37 public static Group[] groups;
38 
39 /**
40  * The array of Item unit of measurements (UMs)
41  */
42 public static UM[] umArray;
43 
44 /**
45  * The available locations in the warehouse
46  */
47 public static WarehouseLocations locations;
48 
49 /**
50  * The user that logged in
51  */
52 private User user;
53 
54 /**
55  * The Main Screen's contentPane
56  */
57 public static Container contentPane;
58 
59 /**
60  * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
61  */
62 {
63     //Set Look & Feel
64     try {
65         javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
66     } catch(Exception e) {
67         e.printStackTrace();
68     }
69 }
70 }
71 
72 /**
```

```
73     * GUI components generated by Jigloo
74     */
75
76     private WarehouseMenuBar mnuMain;
77     private JPanel pnlMainScreen;
78     private WarehouseToolbar tlbMain;
79
80     /**
81      * Constructs a new MainScreen object
82      * @param user - the user that logged in
83      */
84     public MainScreen(User user) {
85         super();
86         this.user = user;
87         this.contentPane = getContentPane();
88         populateUtilities();
89         initGUI(user);
90     }
91
92     /**
93      * Method generated by Jigloo to initialize GUI components
94      */
95     private void initGUI(User user) {
96         try {
97             this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
98             this.setTitle("Warehouse Application");
99             this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
100                 "\\\Icons\\\\WarehouseLogo.png")).getImage());
101            this.setResizable(false);
102            this.setFocusTraversalKeysEnabled(false);
103            this.setLocation(new java.awt.Point(100, 100));
104            {
105                ChangeScreen.setBlankScreen(user);
106            }
107            {
108                mnuMain = new WarehouseMenuBar(user, this);
```

```
109         setJMenuBar(mnuMain);
110     }
111     pack();
112     this.setSize(809, 577);
113 } catch (Exception e) {
114     e.printStackTrace();
115 }
116 }
117
118 /**
119 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
120 */
121
122 /**
123 * Gets the user that logged in
124 * @return the user that logged in
125 */
126 public User getUser(){
127     return user;
128 }
129
130 /**
131 * Initializes the application's utilities
132 */
133 public void populateUtilities(){
134     MainScreen.groups = getGroups(); //Populate Groups
135     MainScreen.umArray = getUMs(); //Populate Units of Measurement (UMs)
136     MainScreen.locations = new WarehouseLocations(); //Populate the Locations
137     MainScreen.codeIndexTree = new IndexBTree();
138     MainScreen.nameIndexTree = new IndexBTree();
139     populateBinaryTrees();
140 }
141
142 /**
143 * Reads the Group file to get an array of Groups
144 * @return an array of Groups
```

```
145 * @mastery achieves HL mastery factor 9 by parsing the Groups text file to obtain an array of Groups
146 */
147 public Group[] getGroups(){
148     try{
149         File file = new File(ApplicationConstants.GROUPS_FILE);
150         FileReader reader = new FileReader(file);
151         BufferedReader buff = new BufferedReader(reader);
152
153         boolean eof = false;           //stores whether the end of the file has been reached
154
155         int numberOfGroups = 0;
156
157         while(!eof){
158             String line = buff.readLine();
159             if (line == null){
160                 eof = true;           //the end of the file has been reached
161             } else {
162                 numberOfGroups++;
163             }
164         }
165
166         Group[] groups = new Group[numberOfGroups];
167
168         reader.close();
169         buff.close();
170
171         FileReader reader2 = new FileReader(file);
172         BufferedReader buff2 = new BufferedReader(reader2);
173
174         for (int i = 0;i<numberOfGroups;i++){
175             String line = buff2.readLine();
176             groups[i] = readGroup(line);
177         }
178
179         return groups;
180     }
```

```
181     } catch (Exception e) {
182         return null;
183     }
184 }
185
186 /**
187 * Returns a Group object from a tokenized String
188 * @param line - a tokenized String
189 * @return a Group object from that String
190 */
191 public Group readGroup(String line){
192     StringTokenizer tokenizer = new StringTokenizer(line, "|");
193     String code = tokenizer.nextToken();
194
195     while(code.startsWith("0")){
196         code = code.substring(1, code.length());
197     }
198
199     short codeValue = new Integer(code).shortValue();
200     String name = tokenizer.nextToken();
201     return new Group(codeValue, name);
202 }
203
204 /**
205 * Reads the Group file to get an array of Groups
206 * @return an array of Groups
207 */
208 public UM[] getUMs() {
209     try{
210         File file = new File(ApplicationConstants.UM_FILE);
211         FileReader reader = new FileReader(file);
212         BufferedReader buff = new BufferedReader(reader);
213
214         boolean eof = false;          //stores whether the end of the file has been reached
215
216         int numberOfGroups = 0;
```

```
217
218     while(!eof){
219         String line = buff.readLine();
220         if (line == null){
221             eof = true;           //the end of the file has been reached
222         } else {
223             numberGroups++;
224         }
225     }
226
227     UM[] umArray = new UM[numberGroups];
228
229     FileReader reader2 = new FileReader(file);
230     BufferedReader buff2 = new BufferedReader(reader2);
231
232     for (int i = 0;i<numberGroups;i++){
233         String line = buff2.readLine();
234         umArray[i] = readUM(line);
235     }
236
237     return umArray;
238
239 } catch(Exception e){
240     return null;
241 }
242 }
243
244 /**
245 * Returns a UM object from a tokenized String
246 * @param line - a tokenized String
247 * @return a UM object from that String
248 */
249 public UM readUM(String line){
250     StringTokenizer tokenizer = new StringTokenizer(line, "|");
251     String code = tokenizer.nextToken();
252 }
```

```
253     while(code.startsWith("0")){
254         code = code.substring(1, code.length());
255     }
256
257     short codeValue = new Integer(code).shortValue();
258     String name = tokenizer.nextToken();
259     return new UM(codeValue, name);
260 }
261
262 /**
263 * Populates the binary trees of Item Indexes by code and name
264 */
265 public void populateBinaryTrees(){
266     Index[] codeIndexes = readIndexes(new File(ApplicationConstants.CODE_INDEX_FILE));
267     Index[] nameIndexes = readIndexes(new File(ApplicationConstants.NAME_INDEX_FILE));
268
269     createIndexBinaryTree(codeIndexes, MainScreen.codeIndexTree);
270     createIndexBinaryTree(nameIndexes, MainScreen.nameIndexTree);
271 }
272
273 /**
274 * Method makes a recursive call to createNameIndexBTree() to make sure that the items
275 * of the ordered array are added in a way so that the binary tree of indexes is properly balanced
276 * @param indexes - the ordered array of either code or name indexes
277 * @param tree - the tree of indexes
278 */
279 public void createIndexBinaryTree(Index[] indexes, IndexBTree tree){
280     createIndexBinaryTree(indexes, tree, 0, indexes.length);
281 }
282
283 /**
284 * Method is called recursively to make sure that the items of the ordered array are
285 * added in a way so that the binary tree of indexes is properly balanced
286 * @param indexes - the ordered array of either code or name indexes
287 * @param tree - the tree of indexes
```

```
289     * @param start - a pointer to the start of the array block
290     * @param finish - a pointer to the end of the array block
291     */
292     private void createIndexBinaryTree(Index[] indexes, IndexBTree tree, int start, int finish) {
293         int mid = ((finish-start)/2) + start;
294         int size = finish - start;
295
296         if(mid > indexes.length-1){
297             return;
298         } else if (size == 0){
299             tree.insertIndex(new IndexTNode(indexes[mid]));
300         } else {
301             tree.insertIndex(new IndexTNode(indexes[mid]));
302             createIndexBinaryTree(indexes, tree, start, mid);
303             createIndexBinaryTree(indexes, tree, mid+1, finish);
304
305         }
306     }
307
308 /**
309 * Returns an ordered array (by field - either code or name) of Indexes
310 * @param file - the Index file
311 * @return an ordered array of Indexes
312 * @mastery achieves SL mastery factor 10 as it is a user defined method with an appropriate object return value
313 */
314 public Index[] readIndexes(File file){
315     try{
316         FileReader reader = new FileReader(file);
317         BufferedReader buff = new BufferedReader(reader);
318
319         boolean eof = false;           //stores if the end of the file (eof) has been reached
320         int numberOfRecords = 0;
321
322         /**
323          * @mastery achieves SL mastery factor 6 with the while loop until the end of the file is reached
324         */
```

```
325     while(!eof){  
326         String line = buff.readLine();  
327  
328         /**  
329          * @mastery achieves SL mastery factor 4 with the if/else selection block to see if eof is true  
330          */  
331         if(line == null){  
332             eof = true;           //the end of the file has been reached  
333         } else {  
334             numberOfRecords++;  
335         }  
336     }  
337  
338     buff.close();  
339  
340     Index[] indexes = new Index[numberOfRecords];  
341  
342     FileReader reader2 = new FileReader(file);  
343     BufferedReader buff2 = new BufferedReader(reader2);  
344  
345     for(int i=0;i<numberOfRecords;i++){  
346         String line = buff2.readLine();  
347         indexes[i] = new Index(line);  
348     }  
349  
350     Arrays.sort(indexes);  
351     buff.close();  
352     reader.close();  
353     buff2.close();  
354     reader2.close();  
355  
356     return indexes;  
357  
358 } catch (Exception e){  
359     return null;  
}
```

361        }  
362  
363  
364        }

```
1 import java.awt.event.*;
2 import javax.swing.*;
3 import java.io.*;
4 import java.util.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * Constructs the Create User screen and handles new user creation
10 * @author Alvaro Morales
11 * @date 10/06/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class ManageUsers extends javax.swing.JPanel {
18
19 /**
20 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
21 */
22
23 /**
24 * GUI components generated by Jigloo
25 */
26 private JLabel lblManageUsers;
27 private JScrollPane scpUsers;
28 private JButton btnDelete;
29 private JButton btnNewUser;
30 private JButton btnEdit;
31 private JTable tblUsers;
32 private UsersTableModel model;
33
34 /**
35 * The user that logged in
36 */
```

```
37  private User user;
38
39  {
40      //Set Look & Feel
41      try {
42          javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
43      } catch(Exception e) {
44          e.printStackTrace();
45      }
46  }
47
48 /**
49 * Constructs a new ManageUsers object
50 */
51 public ManageUsers(User user) {
52     super();
53     this.user = user;
54     initGUI();
55 }
56
57 /**
58 * Method generated by Jigloo to initialize GUI components
59 */
60 private void initGUI() {
61     try {
62         this.setPreferredSize(new java.awt.Dimension(800, 500));
63         this.setSize(800, 500);
64         this.setLayout(null);
65     {
66         lblManageUsers = new JLabel();
67         this.add(lblManageUsers);
68         lblManageUsers.setText("Manage Users");
69         lblManageUsers.setBounds(12, 12, 104, 19);
70         lblManageUsers.setFont(new java.awt.Font("Dialog",1,14));
71     }
72 }
```

```
73     scpUsers = new JScrollPane();
74     this.add(scpUsers);
75     scpUsers.setBounds(24, 62, 754, 349);
76     {
77         tblUsers = new JTable();
78         scpUsers.setViewportView(tblUsers);
79         tblUsers.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
80         this.model = new UsersTableModel(getAllUsers());
81         tblUsers.setModel(model);
82     }
83 }
84 {
85     btnEdit = new JButton();
86     this.add(btnEdit);
87     btnEdit.setText("Edit");
88     btnEdit.setBounds(476, 443, 62, 26);
89     btnEdit.addActionListener(new ActionListener() {
90         public void actionPerformed (ActionEvent e){
91             openEditUserFrame();
92         }
93     });
94 }
95 {
96     btnNewUser = new JButton();
97     this.add(btnNewUser);
98     btnNewUser.setText("New User");
99     btnNewUser.setBounds(288, 443, 89, 26);
100    btnNewUser.addActionListener(new ActionListener() {
101        public void actionPerformed (ActionEvent e){
102            openNewUserFrame();
103        }
104    });
105 }
106 {
107     btnDelete = new JButton();
108     this.add(btnDelete);
```

```
109         btnDelete.setText("Delete");
110         btnDelete.setBounds(388, 443, 70, 26);
111         btnDelete.addActionListener(new ActionListener() {
112             public void actionPerformed (ActionEvent e){
113                 deleteUser();
114             }
115         });
116     }
117 } catch (Exception e) {
118     e.printStackTrace();
119 }
120 }
121 /**
122 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
123 */
124
125 /**
126 *
127 * Creates a new NewUser() object and sets it visible.
128 * This method is called when the New User JButton is pressed
129 */
130 public void openNewUserFrame(){
131     NewUser newUser = new NewUser(user);
132     newUser.setLocation(180, 60);
133     newUser.setVisible(true);
134     this.setEnabled(false);
135 }
136 /**
137 * Creates a new EditUser() object and sets it visible.
138 * This method is called when the Edit User JButton is pressed
139 */
140 public void openEditUserFrame(){
141
142     if(tblUsers.getSelectedRow()!= -1){
143         EditUser editUser = new EditUser(model.getAllUsers()[tblUsers.getSelectedRow()]);
144 }
```

```
145         editUser.setLocation(180, 60);
146         editUser.setVisible(true);
147         model.fireTableRowsUpdated(tblUsers.getSelectedRow(), tblUsers.getSelectedRow());
148     }
149
150 }
151
152 /**
153 * Gets a user from a selected row and calls the performDelete() method on it
154 */
155 public void deleteUser(){
156
157     if(tblUsers.getSelectedRow() != -1){
158         performDelete(model.getAllUsers()[tblUsers.getSelectedRow()]);
159         ChangeScreen.setManageUsersScreen(user);
160     }
161 }
162
163 /**
164 * Deletes a user from the Users file by writting a temporary file without the user and then renaming
165 */
166 public void performDelete(User user) {
167     try{
168         File file = new File(ApplicationConstants.USERS_FILE);
169         File tmpFile = new File(ApplicationConstants.USERS_TMP_FILE);           //Creates the temporary file
170         FileReader reader = new FileReader(file);
171         BufferedReader buff = new BufferedReader(reader);
172         FileWriter writer = new FileWriter(tmpFile);
173         BufferedWriter buffwriter = new BufferedWriter(writer);
174
175         boolean eof = false;          //stores if the end of the file (eof) has been reached
176
177         while(!eof){
178             String line = buff.readLine();
179             if(line == null){
180                 eof = true;           //the end of the file has been reached
```

```
181     } else {
182         User userInFile = readUser(line);
183
184         if (!(userInFile.getUsername().equals(user.getUsername()))){
185             buffwriter.write(userInFile.toString());
186             buffwriter.newLine();
187         }
188     }
189
190
191     }
192
193     buffwriter.close();
194     buff.close();
195
196
197     //Rename temporary file to old file
198     file.delete();
199     tmpFile.renameTo(file);
200
201     JOptionPane.showMessageDialog(this, "The user has been successfully deleted",
202         "Information", JOptionPane.INFORMATION_MESSAGE);
203 } catch (Exception e){
204     JOptionPane.showMessageDialog(this, "The user has not been deleted. \n"
205         + The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);
206 }
207 }
208
209 /**
210 * Gets all the users from the file to populate the table
211 * @return a User[] of all the users from the file
212 */
213 public User[] getAllUsers(){
214     try{
215         File file = new File(ApplicationConstants.USERS_FILE);
216         FileReader reader = new FileReader(file);
```

```
217    BufferedReader buff = new BufferedReader(reader);  
218  
219    boolean eof = false;           //stores whether the end of the file has been reached  
220  
221    int numberOfUsers = 0;  
222  
223    while(!eof){  
224        String line = buff.readLine();  
225        if (line == null){  
226            eof = true;           //the end of the file has been reached  
227        } else {  
228            numberOfUsers++;  
229        }  
230    }  
231  
232    buff.close();  
233    reader.close();  
234  
235    FileReader reader2 = new FileReader(file);  
236    BufferedReader buff2 = new BufferedReader(reader2);  
237  
238    User[] users = new User[numberOfUsers];  
239  
240    for(int i=0;i<numberOfUsers;i++){  
241        String line = buff2.readLine();  
242        users[i] = readUser(line);  
243    }  
244  
245    buff2.close();  
246    reader2.close();  
247  
248    return users;  
249  
250 } catch(Exception e){  
251     return null;  
252 }
```

```
253     }
254
255     /**
256      * Creates a User object from a tokenized String input
257      * @param line - a line read from the Users file
258      * @return - a User object
259      */
260     public User readUser(String line){
261         StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
262
263         if (tokenizer.countTokens() == 7){          //preliminary error checking
264             String username = tokenizer.nextToken();
265             String password = tokenizer.nextToken();
266             String fullName = tokenizer.nextToken();
267             boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
268             boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
269             boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
270             boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
271
272             return new User(username, password, fullName, admin, entry, exit, userEnabled);
273         }
274
275         return null;
276     }
277
278     /**
279      * Converts an int to a boolean
280      * @param 0 if false, 1 if true
281      * @return true if param is 1, else false
282      */
283     public boolean intToBoolean(int i){
284         return (i==1);
285     }
286
287 }
288 }
```

```
1 import java.awt.event.*;
2 import java.util.ArrayList;
3 import javax.swing.event.*;
4 import javax.swing.*;
5 import javax.swing.border.*;
6 import javax.swing.table.*;
7 import java.awt.*;
8
9 /**
10 * -----
11 * Warehouse Application
12 * The Multiple Item Search screen to find multiple items
13 * @author Alvaro Morales
14 * @date 10/07/2010
15 * @school Markham College
16 * @IDE Eclipse SDK
17 * @computer IBM ThinkPad R52
18 * -----
19 */
20 public class MultipleItemSearch extends javax.swing.JPanel {
21
22     /**
23      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
24      */
25
26     {
27         //Set Look & Feel
28         try {
29             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     /**
36      * GUI components generated by Jigloo
```

```
37     */
38     private JButton btnAdd;
39     private JLabel lblRow;
40     private JLabel lblAisle;
41     private JSeparator spSearch;
42     private JComboBox cmbColumn;
43     private JComboBox cmbRow;
44     private JComboBox cmbAisle;
45     private JLabel lblLocation;
46     private JLabel lblSearchResultsInfo;
47     private JButton btnNextStep;
48     private JButton btnSearch;
49     private JTextField txtSearchByName;
50     private JLabel lblCodeSearchInfo;
51     private JRadioButton rbtnExactMatch;
52     private JRadioButton rbtnPartialMatch;
53     private ButtonGroup btnGroupSearchType;
54     private JComboBox cmbGroup;
55     private JLabel lblGroup;
56     private JLabel lblName;
57     private JLabel lblColumn;
58     private JButton btnRemove;
59     private JList lstItems;
60     private JLabel lblItemListInfo;
61     private JLabel lblItemList;
62     private JTable tblSearchResults;
63     private JLabel lblMultipleItemSearch;
64     private JSeparator spSearchList;
65     private JScrollPane scpSearchResults;
66     private JTextField txtSearchByCode;
67     private JLabel lblCode;
68     private JComboBox cmbSearchCriteria;
69     private JLabel lblSearchBy;
70     private JLabel lblItemSearchInfo;
71
72     /**

```

```
73     * The user that logged in
74     */
75     private User user;
76
77     /**
78      * An ArrayList of items to process
79      */
80     private ArrayList itemsToProcess;
81
82     /**
83      * The type of transaction
84      */
85     private int transactionType;
86
87     /**
88      * Constructs a new MultipleItemSearch object
89      * @param user - the user that logged in
90      * @param transactionType - type of transaction (Item entry or exit)
91      */
92     public MultipleItemSearch(User user, int transactionType) {
93         super();
94         this.user = user;
95         this.transactionType = transactionType;
96         initGUI();
97     }
98
99     /**
100      * Method generated by Jigloo to initialize GUI components
101      */
102     private void initGUI() {
103         try {
104             this.setPreferredSize(new java.awt.Dimension(800, 500));
105             this.setSize(800, 500);
106             this.setLayout(null);
107             {
108                 lblItemSearchInfo = new JLabel();
```

```
109     this.add(lblItemSearchInfo);
110     lblItemSearchInfo.setText("Select an item and click 'add'");
111     lblItemSearchInfo.setBounds(12, 30, 254, 16);
112     lblItemSearchInfo.setFont(new java.awt.Font("Dialog",0,10));
113 }
114 {
115     lblMultipleItemSearch = new JLabel();
116     this.add(lblMultipleItemSearch);
117     lblMultipleItemSearch.setText("1. Multiple Item Search");
118     lblMultipleItemSearch.setFont(new java.awt.Font("Dialog",1,14));
119     lblMultipleItemSearch.setBounds(12, 10, 171, 16);
120 }
121 {
122     lblSearchBy = new JLabel();
123     this.add(lblSearchBy);
124     lblSearchBy.setText("Search by:");
125     lblSearchBy.setBounds(35, 58, 63, 19);
126 }
127 {
128     ComboBoxModel cmbSearchByModel =
129         new DefaultComboBoxModel(
130             new String[] { "Code", "Name", "Group", "Location" });
131     cmbSearchCriteria = new JComboBox();
132     this.add(cmbSearchCriteria);
133     cmbSearchCriteria.setModel(cmbSearchByModel);
134     cmbSearchCriteria.setBounds(110, 55, 179, 25);
135     cmbSearchCriteria.addActionListener(new ActionListener() {
136         public void actionPerformed (ActionEvent e){
137             changeSearchCriteria();
138         }
139     });
140 }
141 {
142     lblCode = new JLabel();
143     this.add(lblCode);
144     lblCode.setText("Code:");
```

```
145     lblCode.setBounds(61, 95, 32, 16);
146 }
147 {
148     txtSearchByCode = new JTextField();
149     this.add(txtSearchByCode);
150     txtSearchByCode.setBounds(110, 91, 220, 24);
151     txtSearchByCode.setSize(220, 24);
152     txtSearchByCode.setDocument(new TextFieldDigitLimit(6));
153 }
154 {
155     scpSearchResults = new JScrollPane();
156     this.add(scpSearchResults);
157     scpSearchResults.setBounds(35, 213, 415, 232);
158 {
159     MultipleSearchResultsTableModel model = new MultipleSearchResultsTableModel(null);
160     tblSearchResults = new JTable();
161     tblSearchResults.setModel(model);
162     scpSearchResults.setViewportView(tblSearchResults);
163     tblSearchResults.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
164     tblSearchResults.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
165     setTableColumnsWidth();
166     tblSearchResults.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
167         public void valueChanged(ListSelectionEvent e) {
168             enableAdd(e.getFirstIndex());
169         }
170     });
171     tblSearchResults.addMouseListener(new MouseAdapter() {
172         public void mouseClicked(MouseEvent e) {
173             if (e.getClickCount() == 2) {
174                 JTable tblResults = (JTable)e.getSource();
175                 int row = tblResults.getSelectedRow();
176                 MultipleSearchResultsTableModel model =
177                     (MultipleSearchResultsTableModel)tblSearchResults.getModel();
178                 Item item = (Item)model.getSearchResults().get(row);
179                 ItemDescription description = new ItemDescription(item, user);
180                 description.setVisible(true);
```

```
181
182
183
184
185
186
187
188
189
190     spSearchList = new JSeparator();
191     this.add(spSearchList);
192     spSearchList.setBounds(481, 12, 2, 471);
193     spSearchList.setOrientation(SwingConstants.VERTICAL);
194
195
196     itemsToProcess = new ArrayList();
197     btnAdd = new JButton();
198     this.add(btnAdd);
199     btnAdd.setText("Add");
200     btnAdd.setBounds(378, 457, 72, 26);
201     btnAdd.setEnabled(false);
202     btnAdd.addActionListener(new ActionListener() {
203         public void actionPerformed(ActionEvent e){
204             addItemToList((DefaultListModel)lstItems.getModel());
205         }
206     });
207
208
209     lblItemList = new JLabel();
210     this.add(lblItemList);
211     lblItemList.setText("2. Item List");
212     lblItemList.setFont(new java.awt.Font("Dialog",1,14));
213     lblItemList.setBounds(501, 10, 101, 16);
214
215
216     lblItemListInfo = new JLabel();
```

```
217     this.add(lblItemListInfo);
218     lblItemListInfo.setText("To remove an item, select it and click 'remove'");
219     lblItemListInfo.setFont(new java.awt.Font("Dialog",0,10));
220     lblItemListInfo.setBounds(501, 30, 256, 16);
221 }
222 {
223     DefaultListModel lstItemsModel = new DefaultListModel();
224     lstItems = new JList();
225     this.add(lstItems);
226     lstItems.setModel(lstItemsModel);
227     lstItems.setBounds(513, 58, 255, 240);
228     lstItems.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
229     lstItems.setSize(255, 247);
230     lstItems.setAutoscrolls(false);
231     lstItems.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
232         public void valueChanged(ListSelectionEvent e) {
233             enableRemove(e.getFirstIndex());
234         }
235     });
236 }
237 {
238     btnRemove = new JButton();
239     this.add(btnRemove);
240     btnRemove.setText("Remove");
241     btnRemove.setEnabled(false);
242     btnRemove.setBounds(689, 317, 80, 26);
243     btnRemove.addActionListener(new ActionListener() {
244         public void actionPerformed (ActionEvent e){
245             removeItemFromList();
246             if(lstItems.getModel().getSize() == 0){
247                 btnRemove.setEnabled(false);
248             }
249         }
250     });
251 }
252 {
```

```
253     btnSearch = new JButton();
254     this.add(btnSearch);
255     btnSearch.setText("Search");
256     btnSearch.setBounds(110, 141, 92, 26);
257     btnSearch.addActionListener(new ActionListener() {
258         public void actionPerformed (ActionEvent e){
259             performSearch();
260         }
261     });
262 }
263 {
264     btnNextStep = new JButton();
265     this.add(btnNextStep);
266     btnNextStep.setText("Next Step");
267     btnNextStep.setBounds(689, 457, 89, 26);
268     btnNextStep.addActionListener(new ActionListener() {
269         public void actionPerformed(ActionEvent e){
270             proceedToProcessing();
271         }
272     });
273 }
274 {
275     lblSearchResultsInfo = new JLabel();
276     this.add(lblSearchResultsInfo);
277     lblSearchResultsInfo.setText("Double click on an item to view more information");
278     lblSearchResultsInfo.setFont(new java.awt.Font("Dialog",0,10));
279     lblSearchResultsInfo.setBounds(35, 185, 275, 16);
280 }
281 {
282     lblLocation = new JLabel();
283     this.add(lblLocation);
284     lblLocation.setText("Location:");
285     lblLocation.setBounds(43, 95, 52, 16);
286     lblLocation.setVisible(false);
287 }
288 {
```

```
289     ComboBoxModel cmbAisleModel = new DefaultComboBoxModel(MainScreen.locations.getAisles());
290     cmbAisle = new JComboBox();
291     this.add(cmbAisle);
292     cmbAisle.setModel(cmbAisleModel);
293     cmbAisle.setBounds(109, 91, 52, 25);
294     cmbAisle.setVisible(false);
295 }
296 {
297     ComboBoxModel cmbRowModel = new DefaultComboBoxModel(MainScreen.locations.getRows());
298     cmbRow = new JComboBox();
299     this.add(cmbRow);
300     cmbRow.setModel(cmbRowModel);
301     cmbRow.setBounds(171, 91, 52, 25);
302     cmbRow.setVisible(false);
303 }
304 {
305     ComboBoxModel cmbColumnModel = new DefaultComboBoxModel(MainScreen.locations.getColumns());
306     cmbColumn = new JComboBox();
307     this.add(cmbColumn);
308     cmbColumn.setModel(cmbColumnModel);
309     cmbColumn.setBounds(234, 93, 52, 25);
310     cmbColumn.setVisible(false);
311 }
312 {
313     spSearch = new JSeparator();
314     this.add(spSearch);
315     spSearch.setBounds(12, 177, 452, 10);
316 }
317 {
318     lblAisle = new JLabel();
319     this.add(lblAisle);
320     lblAisle.setText("Aisle");
321     lblAisle.setBounds(125, 117, 25, 16);
322     lblAisle.setFont(new java.awt.Font("Dialog", 0, 10));
323     lblAisle.setVisible(false);
324 }
```

```
325 {
326     lblRow = new JLabel();
327     this.add(lblRow);
328     lblRow.setText("Row");
329     lblRow.setBounds(183, 117, 25, 16);
330     lblRow.setFont(new java.awt.Font("Dialog",0,10));
331     lblRow.setVisible(false);
332 }
333 {
334     lblColumn = new JLabel();
335     this.add(lblColumn);
336     lblColumn.setText("Column");
337     lblColumn.setBounds(241, 118, 39, 16);
338     lblColumn.setFont(new java.awt.Font("Dialog",0,10));
339     lblColumn.setVisible(false);
340 }
341 {
342     lblName = new JLabel();
343     this.add(lblName);
344     lblName.setText("Name:");
345     lblName.setBounds(57, 95, 36, 16);
346     lblName.setVisible(false);
347 }
348 {
349     lblGroup = new JLabel();
350     this.add(lblGroup);
351     lblGroup.setText("Group:");
352     lblGroup.setBounds(56, 95, 37, 16);
353     lblGroup.setVisible(false);
354 }
355 {
356     ComboBoxModel cmbGroupModel = new DefaultComboBoxModel(MainScreen.groups);
357     cmbGroup = new JComboBox();
358     this.add(cmbGroup);
359     cmbGroup.setModel(cmbGroupModel);
360     cmbGroup.setBounds(108, 93, 179, 25);
```

```
361         cmbGroup.setVisible(false);
362     }
363     {
364         rbtnPartialMatch = new JRadioButton();
365         rbtnPartialMatch.setText("Partial Match");
366         rbtnPartialMatch.setBounds(346, 64, 106, 24);
367         rbtnPartialMatch.setVisible(false);
368         this.add(rbtnPartialMatch);
369         rbtnPartialMatch.setSelected(true);
370         rbtnPartialMatch.setVisible(false);
371     }
372     {
373         rbtnExactMatch = new JRadioButton();
374         rbtnExactMatch.setText("Exact Match");
375         rbtnExactMatch.setBounds(346, 91, 95, 24);
376         this.add(rbtnExactMatch);
377         rbtnExactMatch.setVisible(false);
378     }
379     {
380         btnGroupSearchType = new ButtonGroup();
381         btnGroupSearchType.add(rbtnPartialMatch);
382         btnGroupSearchType.add(rbtnExactMatch);
383     }
384     {
385         lblCodeSearchInfo = new JLabel();
386         lblCodeSearchInfo.setText("Only digits");
387         lblCodeSearchInfo.setBounds(337, 95, 59, 16);
388         lblCodeSearchInfo.setFont(new java.awt.Font("Dialog", 0, 10));
389         this.add(lblCodeSearchInfo);
390         lblCodeSearchInfo.setVisible(true);
391     }
392     {
393         txtSearchByName = new JTextField();
394         txtSearchByName.setBounds(111, 91, 220, 24);
395         this.add(txtSearchByName);
396         txtSearchByName.setDocument(new TextFieldLimit(200));
```

```
397             txtSearchByName.setVisible(false);
398         }
399     } catch (Exception e) {
400         e.printStackTrace();
401     }
402 }
403
404 /**
405 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
406 */
407
408 /**
409 * Updates GUI elements to match the search criteria and type
410 */
411 public void changeSearchCriteria(){
412     String searchCriteria = (String)cmbSearchCriteria.getSelectedItem();
413
414     if (searchCriteria.equals("Code")) {
415         //Enable and display GUI elements for a search by code
416         lblCode.setVisible(true);
417         txtSearchByCode.setVisible(true);
418         txtSearchByCode.setDocument(new TextFieldDigitLimit(6));
419         lblCodeSearchInfo.setVisible(true);
420
421         //Disable and make invisible GUI elements for other types of searches
422         lblName.setVisible(false);
423         txtSearchByName.setVisible(false);
424         rbtnPartialMatch.setVisible(false);
425         rbtnExactMatch.setVisible(false);
426
427         lblGroup.setVisible(false);
428         cmbGroup.setVisible(false);
429
430         lblLocation.setVisible(false);
431         lblAisle.setVisible(false);
432         lblRow.setVisible(false);
```

```
433     lblColumn.setVisible(false);
434     cmbAisle.setVisible(false);
435     cmbRow.setVisible(false);
436     cmbColumn.setVisible(false);
437
438     this.validate();
439 } else if (searchCriteria.equals("Name")){
440     //Enable and display GUI elements for a search by name
441     lblName.setVisible(true);
442     txtSearchByName.setVisible(true);
443     txtSearchByName.setDocument(new TextFieldLimit(200));
444     rbtnPartialMatch.setVisible(true);
445     rbtnExactMatch.setVisible(true);
446
447     //Disable and make invisible GUI elements for other types of searches
448     lblCode.setVisible(false);
449     txtSearchByCode.setVisible(false);
450     lblCodeSearchInfo.setVisible(false);
451
452     lblGroup.setVisible(false);
453     cmbGroup.setVisible(false);
454
455     lblLocation.setVisible(false);
456     lblAisle.setVisible(false);
457     lblRow.setVisible(false);
458     lblColumn.setVisible(false);
459     cmbAisle.setVisible(false);
460     cmbRow.setVisible(false);
461     cmbColumn.setVisible(false);
462
463     this.validate();
464 } else if (searchCriteria.equals("Group")){
465     //Enable and display GUI elements for a search by group
466     lblGroup.setVisible(true);
467     cmbGroup.setVisible(true);
468
```

```
469 //Disable and make invisible GUI elements for other types of searches
470 lblCode.setVisible(false);
471 txtSearchByCode.setVisible(false);
472 lblCodeSearchInfo.setVisible(false);
473
474 lblName.setVisible(false);
475 txtSearchByName.setVisible(false);
476 rbtnPartialMatch.setVisible(false);
477 rbtnExactMatch.setVisible(false);
478
479 lblLocation.setVisible(false);
480 lblAisle.setVisible(false);
481 lblRow.setVisible(false);
482 lblColumn.setVisible(false);
483 cmbAisle.setVisible(false);
484 cmbRow.setVisible(false);
485 cmbColumn.setVisible(false);
486
487 this.validate();
488 } else if(searchCriteria.equals("Location")){
489 //Enable and display GUI elements for a search by location
490 lblLocation.setVisible(true);
491 lblAisle.setVisible(true);
492 lblRow.setVisible(true);
493 lblColumn.setVisible(true);
494 cmbAisle.setVisible(true);
495 cmbRow.setVisible(true);
496 cmbColumn.setVisible(true);
497
498 //Disable and make invisible GUI elements for other types of searches
499 lblCode.setVisible(false);
500 txtSearchByCode.setVisible(false);
501 lblCodeSearchInfo.setVisible(false);
502
503 lblName.setVisible(false);
504 txtSearchByName.setVisible(false);
```

```
505         rbtnPartialMatch.setVisible(false);
506         rbtnExactMatch.setVisible(false);
507
508         lblGroup.setVisible(false);
509         cmbGroup.setVisible(false);
510
511         this.validate();
512     }
513 }
514
515 /**
516 * Performs a search by code on the Code Index Binary Tree.
517 * Updates the table to show the items matching.
518 * If there is no match, the user is alerted.
519 * @param codeToFind - the code of the item to find (the search query)
520 */
521 public void performSearchByCode(String codeToFind) {
522
523     Item item = MainScreen.codeIndexTree.search(codeToFind);
524
525     if(item == null){
526         displayNoResultsMessage();
527         txtSearchByCode.setText("");
528     } else {
529         ArrayList list = new ArrayList();
530         list.add(item);
531         list.trimToSize();
532         tblSearchResults.setModel(new MultipleSearchResultsTableModel(list));
533         tblSearchResults.validate();
534         setTableColumnsWidth();
535     }
536
537 }
538
539 /**
540 * Performs a search by exact name on the Name Index Binary Tree.
```

```
541     * Updates the table to show the items matching.  
542     * If there is no match, the user is alerted.  
543     * @param exactName - the exact name of the item to find (the search query)  
544     */  
545     public void performExactSearchByName(String exactName){  
546         Item item = MainScreen.nameIndexTree.search(exactName);  
547  
548         if(item == null){  
549             displayNoResultsMessage();  
550             txtSearchByName.setText("");  
551         } else {  
552             ArrayList list = new ArrayList();  
553             list.add(item);  
554             list.trimToSize();  
555             tblSearchResults.setModel(new MultipleSearchResultsTableModel(list));  
556             tblSearchResults.validate();  
557             setTableColumnsWidth();  
558         }  
559     }  
560  
561     /**  
562      * Enables the add button  
563      * @param i - the value from a row selection (-1 if no row is selected)  
564      */  
565     public void enableAdd(int i){  
566         if(i<0){  
567             btnAdd.setEnabled(false);  
568         } else {  
569             btnAdd.setEnabled(true);  
570         }  
571     }  
572  
573     /**  
574      * Performs a search by partial name on the Name Index Binary Tree.  
575      * @param partialName - the partial match of the name of the item to find (the search query)  
576      * Updates the table to show the items matching.
```

```
577 * If there is no match, the user is alerted.  
578 */  
579 public void performPartialSearchByName(String partialName){  
580     ArrayList searchResults = MainScreen.nameIndexTree.partialSearch(partialName);  
581  
582     if(searchResults == null){  
583         displayNoResultsMessage();  
584         txtSearchByName.setText("");  
585     } else {  
586         tblSearchResults.setModel(new MultipleSearchResultsTableModel(searchResults));  
587         tblSearchResults.validate();  
588         setTableColumnsWidth();  
589     }  
590 }  
591  
592 /**  
593 * Alerts the user that no results matching the query have been found and clears the table  
594 */  
595 public void displayNoResultsMessage(){  
596     JOptionPane.showMessageDialog(this, "No results", "Error", JOptionPane.ERROR_MESSAGE);  
597     tblSearchResults.setModel(new MultipleSearchResultsTableModel(null));  
598     tblSearchResults.validate();  
599     setTableColumnsWidth();  
600 }  
601  
602 /**  
603 * Performs a search depending on the criteria selected  
604 */  
605 public void performSearch(){  
606     if (txtSearchByCode.isVisible()) {  
607         if(!txtSearchByCode.getText().equals("")){  
608             if(txtSearchByCode.getText().length() != 6){  
609                 JOptionPane.showMessageDialog(this, "You must enter a 6 digit code query",  
610                     "Error", JOptionPane.ERROR_MESSAGE);  
611             } else {  
612                 performSearchByCode(txtSearchByCode.getText());  
613             }  
614         }  
615     }  
616 }
```

```
613         }
614     } else {
615         JOptionPane.showMessageDialog(this, "You must enter a search query",
616             "Error", JOptionPane.ERROR_MESSAGE);
617     }
618 } else if (txtSearchByName.isVisible()){
619     if(!txtSearchByName.getText().equals("")){
620         if(rbtnExactMatch.isSelected()){
621             performExactSearchByName(txtSearchByName.getText());
622         } else {
623             performPartialSearchByName(txtSearchByName.getText());
624         }
625     } else {
626         JOptionPane.showMessageDialog(this, "You must enter a search query",
627             "Error", JOptionPane.ERROR_MESSAGE);
628     }
629 } else if (cmbGroup.isVisible()){
630     Group group = (Group)cmbGroup.getSelectedItem();
631     performSearchByGroup(group.getCode());
632 } else if (cmbAisle.isVisible()){
633     performSearchByLocation();
634 }
635 }
636
637 /**
638 * Performs a search by group, updating the table with the search results
639 * @param groupCode - the code of the group (search query)
640 */
641 public void performSearchByGroup(short groupCode){
642     ArrayList searchResults = Items.searchByGroup(groupCode);
643     if(!searchResults.isEmpty()){
644         tblSearchResults.setModel(new MultipleSearchResultsTableModel(searchResults));
645         tblSearchResults.validate();
646         setTableColumnsWidth();
647     } else {
648         displayNoResultsMessage();
```

```
649         }
650     }
651
652
653     /**
654      * Performs a search by group, updating the table with the search results
655      * @param groupCode - the code of the group (search query)
656     */
657     public void performSearchByLocation(){
658         byte warehouse = 1;
659         byte aisle = ((Integer)cmbAisle.getSelectedItem()).byteValue();
660         char row = ((String)cmbRow.getSelectedItem()).charAt(0);
661         byte column = ((Integer)cmbColumn.getSelectedItem()).byteValue();
662
663         Location location = new Location(warehouse, aisle, column, row);
664         ArrayList searchResults = Items.searchByLocation(location);
665         if(!searchResults.isEmpty() || searchResults == null){
666             tblSearchResults.setModel(new MultipleSearchResultsTableModel(searchResults));
667             tblSearchResults.validate();
668             setTableColumnsWidth();
669         } else {
670             displayNoResultsMessage();
671         }
672     }
673
674     /**
675      * Adds a selected item to the Items list if the item is not already in the list
676     */
677     public void addItemAtList(DefaultListModel listModel){
678         int row = tblSearchResults.getSelectedRow();
679         MultipleSearchResultsTableModel model = (MultipleSearchResultsTableModel)tblSearchResults.getModel();
680         Item item = (Item) model.getSearchResults().get(row); //The item to add
681
682         boolean isRepeated = false;
683
684         for(int i=0; i<lstItems.getModel().getSize(); i++) {
```

```
685     Item currentItem = (Item)lstItems.getModel().getElementAt(i);
686     if(currentItem.getID() == item.getID()){
687         isRepeated = true;
688     }
689 }
690
691 if(transactionType == ApplicationConstants.ITEM_EXIT){
692     int quantity = ((Integer)model.getValueAt(row, 3)).intValue();
693     if(!isRepeated && quantity > 0){
694         listModel.addElement(item);
695         lstItems.setModel(listModel);
696         lstItems.validate();
697     } else if (quantity <= 0){
698         JOptionPane.showMessageDialog(this, "The item is not in stock", "Error", JOptionPane.ERROR_MESSAGE);
699     }
700 } else {
701     if(!isRepeated){
702         listModel.addElement(item);
703         lstItems.setModel(listModel);
704         lstItems.validate();
705     }
706 }
707
708 /**
709 * Enables the remove button
710 * @param i - the value from a row selection (-1 if no row is selected)
711 */
712 public void enableRemove(int i){
713     if(i<0){
714         btnRemove.setEnabled(false);
715     } else {
716         btnRemove.setEnabled(true);
717     }
718 }
719 }
720 }
```

```
721
722     /**
723      * Removes a selected Item from the list
724     */
725     public void removeItemFromList(){
726         DefaultListModel listModel = (DefaultListModel)lstItems.getModel();
727         if(lstItems.getSelectedIndex() != -1){
728             listModel.remove(lstItems.getSelectedIndex());
729             lstItems.validate();
730         }
731     }
732
733     /**
734      * Adds the items from the JList to an ArrayList
735     */
736     public void addListItemsToArrayList(){
737         itemsToProcess = new ArrayList();
738         for(int i=0; i<lstItems.getModel().getSize();i++){
739             Item item = (Item)lstItems.getModel().getElementAt(i);
740             itemsToProcess.add(item);
741         }
742
743         itemsToProcess.trimToSize();
744     }
745
746     /**
747      * Loads the next step of the Transaction process
748     */
749     public void proceedToProcessing(){
750         if(lstItems.getModel().getSize() == 0){
751             JOptionPane.showMessageDialog(this, "You must add items to the list",
752                                         "Error", JOptionPane.ERROR_MESSAGE);
753         } else {
754             addListItemsToArrayList();
755             if(transactionType == ApplicationConstants.ITEM_ENTRY){
756                 ItemEntry entry = new ItemEntry(user, itemsToProcess);
```

```
757         entry.setSize(800, 500);
758         MainScreen.contentPane.remove(this);
759         MainScreen.contentPane.add(entry, BorderLayout.CENTER);
760         MainScreen.contentPane.validate();
761     } else {
762         ItemExit exit = new ItemExit(user, itemsToProcess);
763         exit.setSize(800, 500);
764         MainScreen.contentPane.remove(this);
765         MainScreen.contentPane.add(exit, BorderLayout.CENTER);
766         MainScreen.contentPane.validate();
767     }
768 }
769 }
770
771 /**
772 * Sets the width of the columns in the table
773 */
774 public void setTableColumnsWidth(){
775     //Set Column "Code" width
776     TableColumn colCode = tblSearchResults.getColumnModel().getColumn(0);
777     colCode.setPreferredWidth(60);
778
779     //Set Column "Name" width
780     TableColumn colName = tblSearchResults.getColumnModel().getColumn(1);
781     colName.setPreferredWidth(230);
782
783     //Set Column "UM" width
784     TableColumn colUM = tblSearchResults.getColumnModel().getColumn(2);
785     colUM.setPreferredWidth(67);
786
787     //Set Column "In Stock" width
788     TableColumn colInStock = tblSearchResults.getColumnModel().getColumn(3);
789     colInStock.setPreferredWidth(55);
790
791     //Set Column "Group" width
792     TableColumn colGroup = tblSearchResults.getColumnModel().getColumn(4);
```

```
793     colGroup.setPreferredWidth(100);  
794  
795     //Set Column "Location" width  
796     TableColumn colLocation = tblSearchResults.getColumnModel().getColumn(5);  
797     colLocation.setPreferredWidth(77);  
798 }  
799  
800 }
```

```
1 import javax.swing.table.AbstractTableModel;
2 import java.util.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the New User screen and handles new user creation
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15
16 public class MultipleSearchResultsTableModel extends AbstractTableModel {
17
18     private ArrayList searchResults;
19
20     /**
21      * Constructs a SearchResultsTableModel
22      * @param searchResults - an ArrayList of the search results
23      */
24     public MultipleSearchResultsTableModel(ArrayList searchResults) {
25         this.searchResults = searchResults;
26     }
27
28     /**
29      * Returns the number of columns in the table
30      * @return - the number of columns in the table (7)
31      */
32     public int getColumnCount() {
33         return 6;
34     }
35
36     /**
```

```
37     * Returns the number of rows in the table
38     * @return - the number of rows in the table (the number of items in the ArrayList)
39     */
40    public int getRowCount(){
41        if(searchResults!= null){
42            return searchResults.size();
43        } else {
44            return 0;
45        }
46    }
47
48 /**
49 * Returns the object in a cell of the table at a given row-column location
50 * @param row - row number
51 * @param col - column number
52 * @return - the object at that row-column location
53 */
54    public Object getValueAt(int row, int col){
55
56        Item item = (Item)searchResults.get(row);
57
58        switch(col){
59            case 0:
60                return item.getCode();
61            case 1:
62                return item.getName();
63            case 2:
64                return item.getUm().toString();
65            case 3:
66                return new Integer(Transactions.getQuantityInStock(item));
67            case 4:
68                return item.getGroup().toString();
69            case 5:
70                return item.getLocation().toString();
71        default:
```

```
73         return null;
74     }
75 }
76 /**
77 * Returns the name of a column
78 * @param col - the column number
79 * @return the name of a column
80 */
81 public String getColumnName(int col){
82
83     switch(col){
84
85         case 0:
86             return "Code";
87         case 1:
88             return "Name";
89         case 2:
90             return "U.M.";
91         case 3:
92             return "In Stock";
93         case 4:
94             return "Group";
95         case 5:
96             return "Location";
97         default:
98             return null;
99     }
100 }
101 }
102 /**
103 * Returns the class of a column
104 * @param col - the column number
105 * @return the Class of a column
106 */
107
108 */
```

```
109 public Class getColumnClass(int col){  
110     switch(col){  
111         case 0:  
112             return String.class;  
113         case 1:  
114             return String.class;  
115         case 2:  
116             return String.class;  
117         case 3:  
118             return Integer.class;  
119         case 4:  
120             return String.class;  
121         case 5:  
122             return String.class;  
123         default:  
124             return null;  
125     }  
126 }  
127  
128 }  
129  
130 /**  
131 * Returns if a cell is editable  
132 * @return false, regardless of row-column index  
133 */  
134 public boolean isCellEditable(int row, int col){  
135     return false;  
136 }  
137  
138 /**  
139 * Returns an ArrayList of the search results  
140 * @return an ArrayList of the search results  
141 */  
142 public ArrayList getSearchResults(){  
143     return searchResults;
```

145        }  
146  
147        }

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.io.*;
4 import java.util.*;
5
6 /**
7 * -----
8 * Warehouse Application
9 * Constructs the New User screen and handles new user creation
10 * @author Alvaro Morales
11 * @date 10/06/2010
12 * @school Markham College
13 * @IDE Eclipse SDK
14 * @computer IBM ThinkPad R52
15 * -----
16 */
17 public class NewUser extends javax.swing.JFrame {
18
19 /**
20 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
21 */
22
23 /**
24 * GUI components generated by Jigloo
25 */
26 private JLabel lblNewUser;
27 private JLabel lblFullName;
28 private JCheckBox chkAdmin;
29 private JLabel lblPasswordValidation;
30 private JLabel lblPermissionsValidation;
31 private JLabel lblUsernameValidation;
32 private JSeparator spCreate;
33 private JButton btnCreate;
34 private JCheckBox chkExit;
35 private JCheckBox chkEntry;
36 private JTextField txtFullName;
```

```
37     private JTextField txtPassword;
38     private JTextField txtUsername;
39     private JLabel lblPermissions;
40     private JLabel lblPassword;
41     private JLabel lblUsername;
42
43     /**
44      * The user that logged in
45      */
46     private User user;
47
48     {
49         //Set Look & Feel
50         try {
51             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
52         } catch(Exception e) {
53             e.printStackTrace();
54         }
55     }
56
57     /**
58      * Constructs a new NewUser object
59      */
60     public NewUser() {
61         super();
62         this.user = user;
63         initGUI();
64     }
65
66     /**
67      * Method generated by Jigloo to initialize GUI components
68      */
69     private void initGUI() {
70         try {
71             setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
72             this.setTitle("New User");
```

```
73    getContentPane().setLayout(null);
74    {
75        lblNewUser = new JLabel();
76        getContentPane().add(lblNewUser);
77        lblNewUser.setText("Create a new user");
78        lblNewUser.setBounds(12, 12, 139, 16);
79        lblNewUser.setFont(new java.awt.Font("Dialog",1,14));
80    }
81    {
82        lblUsername = new JLabel();
83        getContentPane().add(lblUsername);
84        lblUsername.setText("Username:");
85        lblUsername.setBounds(51, 40, 62, 16);
86    }
87    {
88        lblPassword = new JLabel();
89        getContentPane().add(lblPassword);
90        lblPassword.setText("Password:");
91        lblPassword.setBounds(53, 76, 61, 16);
92    }
93    {
94        lblFullName = new JLabel();
95        getContentPane().add(lblFullName);
96        lblFullName.setText("Full Name:");
97        lblFullName.setBounds(56, 113, 58, 16);
98    }
99    {
100        lblPermissions = new JLabel();
101        getContentPane().add(lblPermissions);
102        lblPermissions.setText("Permissions:");
103        lblPermissions.setBounds(39, 143, 75, 16);
104    }
105    {
106        txtUsername = new JTextField();
107        txtUsername.setDocument(new TextFieldLimit(20));
108        getContentPane().add(txtUsername);
```

```
109         txtUsername.setBounds(127, 40, 250, 20);
110         txtUsername.setSize(200, 20);
111     }
112     {
113         txtPassword = new JTextField();
114         txtPassword.setDocument(new TextFieldLimit(30));
115         getContentPane().add(txtPassword);
116         txtPassword.setBounds(126, 75, 201, 20);
117     }
118     {
119         txtFullName = new JTextField();
120         getContentPane().add(txtFullName);
121         txtFullName.setBounds(126, 112, 200, 20);
122     }
123     {
124         chkAdmin = new JCheckBox();
125         getContentPane().add(chkAdmin);
126         chkAdmin.setText("Admin");
127         chkAdmin.setBounds(128, 160, 61, 24);
128         chkAdmin.addActionListener(new ActionListener() {
129             public void actionPerformed(ActionEvent e){
130                 if(chkAdmin.isSelected()){
131                     chkEntry.setEnabled(false);
132                     chkEntry.setSelected(true);
133                     chkExit.setEnabled(false);
134                     chkExit.setSelected(true);
135                 } else {
136                     chkEntry.setEnabled(true);
137                     chkEntry.setSelected(false);
138                     chkExit.setEnabled(true);
139                     chkExit.setSelected(false);
140                 }
141             }
142         });
143     }
144 }
```

```
145         chkEntry = new JCheckBox();
146         getContentPane().add(chkEntry);
147         chkEntry.setText("Item Entry Operations");
148         chkEntry.setBounds(128, 185, 148, 24);
149     }
150 }
151 {
152     chkExit = new JCheckBox();
153     getContentPane().add(chkExit);
154     chkExit.setText("Item Exit Operations");
155     chkExit.setBounds(128, 211, 140, 24);
156 }
157 {
158     btnCreate = new JButton();
159     getContentPane().add(btnCreate);
160     btnCreate.setText("Create");
161     btnCreate.setBounds(140, 264, 92, 26);
162     btnCreate.addActionListener(new ActionListener() {
163         public void actionPerformed (ActionEvent e){
164             createUser();
165         }
166     });
167 }
168 {
169     spCreate = new JSeparator();
170     getContentPane().add(spCreate);
171     spCreate.setBounds(12, 247, 342, 10);
172 }
173 {
174     lblUsernameValidation = new JLabel();
175     getContentPane().add(lblUsernameValidation);
176     lblUsernameValidation.setText("Max. 20 characters");
177     lblUsernameValidation.setBounds(23, 53, 91, 16);
178     lblUsernameValidation.setFont(new java.awt.Font("Dialog",0,10));
179 }
180 {
```

```
181     lblPermissionsValidation = new JLabel();
182     getContentPane().add(lblPermissionsValidation);
183     lblPermissionsValidation.setText("(at least one must be selected)");
184     lblPermissionsValidation.setBounds(120, 144, 175, 16);
185     lblPermissionsValidation.setFont(new java.awt.Font("Dialog",0,10));
186 }
187 {
188     lblPasswordValidation = new JLabel();
189     getContentPane().add(lblPasswordValidation);
190     lblPasswordValidation.setText("Max. 30 characters");
191     lblPasswordValidation.setBounds(25, 88, 95, 16);
192     lblPasswordValidation.setFont(new java.awt.Font("Dialog",0,10));
193 }
194 {
195     this.setIconImage(new ImageIcon(getClass().getClassLoader().getResource(
196         ApplicationConstants.USERS_ICON)).getImage());
197     pack();
198     this.setSize(374, 329);
199 }
200 } catch (Exception e) {
201     e.printStackTrace();
202 }
203 }
204 /**
205 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
206 */
207 /**
208 * Checks if the information entered in the fields is correct.
209 * If so, the writeNewUser() method is called.
210 * If the information is not correct, the admin user is alerted
211 */
212 public void createUser(){
213     if(txtUsername.getText() != null && txtPassword.getText() !=null && txtFullName.getText() != null
```

```
217     && (chkAdmin.isSelected() || chkEntry.isSelected() || chkExit.isSelected())){  
218     User user = new User(txtUsername.getText(),txtPassword.getText(), txtFullName.getText(),  
219         chkAdmin.isSelected(),chkEntry.isSelected(),chkExit.isSelected(), true);  
220     try{  
221         writeNewUser(user);  
222         JOptionPane.showMessageDialog(this, "The user has been created",  
223             "Information", JOptionPane.INFORMATION_MESSAGE);  
224         this.dispose();  
225         ChangeScreen.setManageUsersScreen(this.user);  
226     } catch(Exception e) {  
227         e.printStackTrace();  
228         JOptionPane.showMessageDialog(this, "The user has not been created. \n"  
229             + "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);  
230         this.dispose();  
231     }  
232     } else {  
233         JOptionPane.showMessageDialog(this, "Some fields are missing or the \n"  
234             + "information entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);  
235     }  
236 }  
237  
238 /**  
 * Writes a new user to the Users file by creating a temporary file and later renaming it  
 * @param user - the user to be created  
 * @mastery achieves HL mastery factor 17 by inserting a new user into the appropriate position  
 * in the ordered Users sequential file without reading the entire file into RAM  
 */  
244 public void writeNewUser(User user) throws IOException {  
245     try{  
246  
247         File file = new File(ApplicationConstants.USERS_FILE);  
248         File tmpFile = new File(ApplicationConstants.USERS_TMP_FILE); //Creates the temporary file  
249  
250         if(!tmpFile.exists()){  
251             tmpFile.createNewFile();  
252         }
```

```
253
254     FileReader reader = new FileReader(file);
255     BufferedReader buff = new BufferedReader(reader);
256     FileWriter writer = new FileWriter(tmpFile);
257     BufferedWriter buffwriter = new BufferedWriter(writer);
258
259     boolean eof = false;           //stores if the end of the file (eof) has been reached
260     boolean written = false;      //stores if the user to enter has been written to the file
261
262     while(!eof){
263         String line = buff.readLine();
264         if(line == null){
265             if(!written){
266                 buffwriter.write(user.toString());
267                 buffwriter.newLine();
268             }
269             eof = true;           //the end of the file has been reached
270         } else {
271             User userInFile = readUser(line);
272             if(userInFile.getUsername().compareTo(user.getUsername()) < 0){
273                 buffwriter.write(userInFile.toString());
274                 buffwriter.newLine();
275             } else if (userInFile.getUsername().compareTo(user.getUsername()) > 0){
276                 if(!written){
277                     buffwriter.write(user.toString());
278                     buffwriter.newLine();
279                     buffwriter.write(userInFile.toString());
280                     buffwriter.newLine();
281                     written = true;
282                 } else {
283                     buffwriter.write(userInFile.toString());
284                     buffwriter.newLine();
285                 }
286             }
287         }
288     }
```

```
289     }
290
291     reader.close();
292     buff.close();
293     buffwriter.close();
294     writer.close();
295
296     file.delete();
297     tmpFile.renameTo(file);
298
299 } catch (Exception e){
300     e.printStackTrace();
301 }
302
303 }
304
305 /**
306 * Creates a User object from a tokenized String input
307 * @param line - a line read from the Users file
308 * @return - a User object
309 */
310 public User readUser(String line){
311     StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
312
313     if (tokenizer.countTokens() == 7){          //preliminary error checking
314         String username = tokenizer.nextToken();
315         String password = tokenizer.nextToken();
316         String fullName = tokenizer.nextToken();
317         boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
318         boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
319         boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
320         boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
321
322         return new User(username, password, fullName, admin, entry, exit, userEnabled);
323     }
324 }
```

```
325         return null;
326     }
327 }
328 /**
329 * Converts an int to a boolean
330 * @param 0 if false, 1 if true
331 * @return true if param is 1, else false
332 */
333 public boolean intToBoolean(int i){
334     return (i==1);
335 }
336 }
337 }
338 }
```

```
1 import java.awt.BorderLayout;
2 import javax.swing.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs a FileChooser to select an item image from disk
8 * @author Alvaro Morales
9 * @date 21/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15 public class OpenFile extends javax.swing.JFrame {
16
17     /**
18      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
19      */
20
21     /**
22      * The FileChooser object
23      */
24     private JFileChooser flcOpenFile;
25
26     {
27         //Set Look & Feel
28         try {
29             javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     /**
36      * Constructs a new OpenFile object
37  }
```

```
37     */
38     public OpenFile() {
39         super();
40         initGUI();
41     }
42
43     /**
44      * Method generated by Jigloo to initialize GUI components
45      */
46     private void initGUI() {
47         try {
48             setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
49             this.setTitle("Select Image");
50             {
51                 flcOpenFile = new JFileChooser();
52                 getContentPane().add(flcOpenFile, BorderLayout.CENTER);
53             }
54             pack();
55         } catch (Exception e) {
56             e.printStackTrace();
57         }
58     }
59
60 }
```

```
1 import javax.swing.*;
2 import javax.swing.table.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import java.util.*;
6
7 /**
8 * -----
9 * Warehouse Application
10 * The Optimal Route screen that shows an appropriate route for item pickup for withdrawal from the warehouse
11 * @author Alvaro Morales
12 * @date 14/07/2010
13 * @school Markham College
14 * @IDE Eclipse SDK
15 * @computer IBM ThinkPad R52
16 * @mastery achieves HL mastery factor 16 with the use of the java.util external library
17 * -----
18 */
19 public class OptimalRoute extends javax.swing.JPanel {
20
21 /**
22 * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
23 */
24
25 /**
26 * GUI components generated by Jigloo
27 */
28 private JLabel lblOptimalRoute;
29 private JScrollPane scpOptimalRoute;
30 private JButton btnFinish;
31 private JTable tblOptimalRoute;
32 private JLabel lblOptimalRouteInfo;
33
34 /**
35 * An ArrayList of items to calculate the optimal pickup route
36 */
```

```
37  private ArrayList itemsToWithdraw;
38
39  /**
40   * The user that logged in
41   */
42  private User user;
43
44  {
45      //Set Look & Feel
46      try {
47          javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
48      } catch(Exception e) {
49          e.printStackTrace();
50      }
51  }
52
53 /**
54  * Constructs a new OptimalRoute object
55  * @param itemsToWithdraw - an ArrayList of items to withdraw from the warehouse
56  * @param user - the user that logged in
57  */
58 public OptimalRoute(ArrayList itemsToWithdraw, User user) {
59     super();
60     this.itemsToWithdraw = itemsToWithdraw;
61     this.user = user;
62     initGUI();
63 }
64
65 /**
66  * Method generated by Jigloo to initialize GUI components
67  */
68 private void initGUI() {
69     try {
70         this.setLayout(null);
71         this.setPreferredSize(new java.awt.Dimension(800, 500));
72     }
```

```
73     lblOptimalRoute = new JLabel();
74     this.add(lblOptimalRoute);
75     lblOptimalRoute.setText("6. Optimal Route");
76     lblOptimalRoute.setFont(new java.awt.Font("Dialog",1,14));
77     lblOptimalRoute.setBounds(12, 12, 125, 16);
78 }
79 {
80     lblOptimalRouteInfo = new JLabel();
81     this.add(lblOptimalRouteInfo);
82     lblOptimalRouteInfo.setText("The column 'Order' denotes the pickup route");
83     lblOptimalRouteInfo.setFont(new java.awt.Font("Dialog",0,10));
84     lblOptimalRouteInfo.setBounds(12, 34, 252, 16);
85 }
86 {
87     scpOptimalRoute = new JScrollPane();
88     this.add(scpOptimalRoute);
89     scpOptimalRoute.setBounds(51, 70, 698, 359);
90     {
91         RouteTableModel tblOptimalRouteModel = new RouteTableModel(getOptimalRoute(itemsToWithdraw));
92         tblOptimalRoute = new JTable();
93         scpOptimalRoute.setViewportView(tblOptimalRoute);
94         tblOptimalRoute.setModel(tblOptimalRouteModel);
95     }
96
97     tblOptimalRoute.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
98
99     //Set Column "Order" width
100    int vColIndex = 0;
101    TableColumn order = tblOptimalRoute.getColumnModel().getColumn(vColIndex);
102    int width = 70;
103    order.setPreferredWidth(width);
104
105    //Set Column "Code" width
106    TableColumn colID = tblOptimalRoute.getColumnModel().getColumn(1);
107    colID.setPreferredWidth(100);
108
```

```
109     //Set column "Name" width
110     TableColumn colName = tblOptimalRoute.getColumnModel().getColumn(2);
111     colName.setPreferredWidth(345);
112
113     //Set Column "UM" width
114     TableColumn colUM = tblOptimalRoute.getColumnModel().getColumn(3);
115     colUM.setPreferredWidth(100);
116
117     //Set Column "Location" width
118     TableColumn colLocation = tblOptimalRoute.getColumnModel().getColumn(4);
119     colLocation.setPreferredWidth(80);
120 }
121 {
122     btnFinish = new JButton();
123     this.add(btnFinish);
124     btnFinish.setText("Finish");
125     btnFinish.setBounds(670, 455, 79, 26);
126     btnFinish.addActionListener(new ActionListener() {
127         public void actionPerformed (ActionEvent e){
128             ChangeScreen.setBlankScreen(user);
129         }
130     });
131 }
132 } catch (Exception e) {
133     e.printStackTrace();
134 }
135 }
136
137 /**
138 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
139 */
140
141 /**
142 * Gets an optimal pickup route for a set of Items based on their location in the warehouse
143 * @param itemsToProcess - an ArrayList of items being withdrawn
144 * @return an ArrayList of an efficient item pickup route
```

```
145     */
146     public ArrayList getOptimalRoute(ArrayList itemsToProcess) {
147         if(itemsToProcess.size() == 1){
148             return itemsToProcess;
149         } else {
150             ArrayList group1 = new ArrayList();
151             ArrayList group2 = new ArrayList();
152
153             for(int i=0;i<itemsToProcess.size();i++){
154                 Item currentItem = (Item)itemsToWithdraw.get(i);
155
156                 if(currentItem.getLocation().getAisle() <= 36){
157                     group1.add(currentItem);
158                 } else {
159                     group2.add(currentItem);
160                 }
161             }
162
163             group1.trimToSize();
164             Collections.sort(group1, Item.LOCATION_ORDER);
165
166             group2.trimToSize();
167             Collections.sort(group2, Item.LOCATION_ORDER);
168
169             ArrayList route = new ArrayList();
170
171             for(int i=0;i<group1.size();i++){
172                 route.add((Item)group1.get(i));
173             }
174
175             for(int i=0;i<group2.size();i++){
176                 route.add((Item)group2.get(i));
177             }
178
179             route.trimToSize();
180             return route;
```

```
181      }
182      }
183      }
184      }
```

```
1 import java.util.ArrayList;
2 import javax.swing.table.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * The table model for the Optimal Route table
8 * @author Alvaro Morales
9 * @date 14/07/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15 public class RouteTableModel extends AbstractTableModel {
16
17     /**
18      * An ArrayList of Items to pick up from the warehouse
19      */
20     private ArrayList itemsToWithdraw;
21
22
23     /**
24      * Constructs a TransactionsTableModel object to display transactions in a JTable
25      * @param itemsToWithdraw - an ArrayList of TransactionRecord objects that will be displayed in the table
26      */
27     public RouteTableModel(ArrayList itemsToWithdraw) {
28         this.itemsToWithdraw = itemsToWithdraw;
29     }
30
31     /**
32      * Returns the number of columns in the table
33      * @return - the number of columns in the table
34      */
35     public int getColumnCount() {
36         return 5;
```

```
37     }
38
39     /**
40      * Returns the number of rows in the table
41      * @return - the number of rows in the table (the number of items in the ArrayList)
42      */
43     public int getRowCount(){
44         if(itemsToWithdraw!= null){
45             return itemsToWithdraw.size();
46         } else {
47             return 0;
48         }
49     }
50
51     /**
52      * Returns the object in a cell of the table at a given row-column location
53      * @param row - row number
54      * @param col - column number
55      * @return - the object at that row-column location
56      */
57     public Object getValueAt(int row, int col){
58
59         Item item = (Item)itemsToWithdraw.get(row);
60
61         switch(col){
62             case 0:
63                 return new Integer(row+1);
64             case 1:
65                 return item.getCode();
66             case 2:
67                 return item.getName();
68             case 3:
69                 return item.getUm().toString();
70             case 4:
71                 return item.getLocation().toString();
72         default:
```

```
73         return null;
74     }
75 }
76 /**
77 * Returns the name of a column
78 * @param col - the column number
79 * @return the name of a column
80 */
81 public String getColumnName(int col){
82
83     switch(col){
84
85         case 0:
86             return "Order";
87         case 1:
88             return "Code";
89         case 2:
90             return "Name";
91         case 3:
92             return "UM";
93         case 4:
94             return "Location";
95     default:
96         return null;
97     }
98 }
99 /**
100 * Returns if a cell is editable
101 * @return a boolean indicating if the cell is editable
102 */
103 public boolean isCellEditable(int row, int col){
104     return false;
105 }
```

```
109     /**
110      * Gets the ArrayList of TransactionRecord objects to display
111      * @return the ArrayList of TransactionRecord objects to display
112      */
113     public ArrayList getItemsToWithdraw() {
114         return itemsToWithdraw;
115     }
116 }
```

```
1 import javax.swing.table.AbstractTableModel;
2 import java.util.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the New User screen and handles new user creation
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15
16 public class SearchResultsTableModel extends AbstractTableModel {
17
18     /**
19      * An ArrayList of search results
20      */
21     private ArrayList searchResults;
22
23     /**
24      * Constructs a SearchResultsTableModel object
25      * @param searchResults - an ArrayList of the search results
26      */
27     public SearchResultsTableModel(ArrayList searchResults) {
28         this.searchResults = searchResults;
29     }
30
31     /**
32      * Returns the number of columns in the table
33      * @return - the number of columns in the table (7)
34      */
35     public int getColumnCount() {
36         return 6;
```

```
37     }
38
39     /**
40      * Returns the number of rows in the table
41      * @return - the number of rows in the table (the number of items in the ArrayList)
42      */
43     public int getRowCount(){
44         if(searchResults!= null){
45             return searchResults.size();
46         } else {
47             return 0;
48         }
49     }
50
51
52     /**
53      * Returns the object in a cell of the table at a given row-column location
54      * @param row - row number
55      * @param col - column number
56      * @return - the object at that row-column location
57      */
58     public Object getValueAt(int row, int col){
59
60         Item item = (Item)searchResults.get(row);
61
62         switch(col){
63             case 0:
64                 return item.getCode();
65             case 1:
66                 return item.getName();
67             case 2:
68                 return item.getGroup().toString();
69             case 3:
70                 return item.getUm().toString();
71             case 4:
72                 return new Integer(T
```

```
73     case 5:
74         return item.getLocation().toString();
75     default:
76         return null;
77     }
78 }
79
80 /**
81 * Returns the name of a column
82 * @param col - the column number
83 * @return the name of a column
84 */
85 public String getColumnName(int col){
86
87     switch(col){
88
89     case 0:
90         return "Code";
91     case 1:
92         return "Name";
93     case 2:
94         return "Group";
95     case 3:
96         return "U.M.";
97     case 4:
98         return "Quantity in Stock";
99     case 5:
100        return "Location";
101    default:
102        return null;
103    }
104 }
105
106 /**
107 * Returns the class of a column
108 * @param col - the column number
```

```
109     * @return the Class of a column
110     */
111    public Class getColumnClass(int col){
112
113        switch(col){
114
115            case 0:
116                return String.class;
117            case 1:
118                return String.class;
119            case 2:
120                return String.class;
121            case 3:
122                return String.class;
123            case 4:
124                return Integer.class;
125            case 5:
126                return String.class;
127            default:
128                return null;
129        }
130    }
131
132    /**
133     * Returns if a cell is editable
134     * @return false, regardless of row-column index
135     */
136    public boolean isCellEditable(int row, int col){
137        return false;
138    }
139
140    /**
141     * Returns an ArrayList of the search results
142     * @return an ArrayList of the search results
143     */
144    public ArrayList getSearchResults()
```

```
145     return searchResults;  
146 }  
147  
148 }
```

```
1 import javax.swing.text.*;
2 
3 /**
4 * -----
5 * Warehouse Application
6 * Limits the maximum number of characters in a JTextField
7 * @author Real Gagnon
8 * @authorsWebsite http://tactika.com/realhome/realhome.html
9 * @date 05/07/1999
10 *
11 */
12 public class TextFieldDigitLimit extends PlainDocument {
13 
14     /**
15      * The maximum number of digits
16      */
17     private int limit;
18 
19     /**
20      * Constructs a new TextFieldDigitLimit object
21      * @param limit - the maximum number of digits
22      */
23     public TextFieldDigitLimit(int limit) {
24         super();
25         this.limit = limit;
26     }
27 
28     /**
29      * Only inserts a character if it is a digit
30      */
31     public void insertString (int offset, String str, AttributeSet attr) throws BadLocationException {
32         if (str == null) return;
33 
34         Character c = new Character(str.charAt(0));
35 
36         if ((getLength() + str.length()) <= limit && Character.isDigit(c.charValue()) == true) {
```

```
37     super.insertString(offset, str, attr);  
38 }  
39 }  
40 }
```

```
1 import javax.swing.text.*;
2 /**
3 * -----
4 * Warehouse Application
5 * Limits the maximum number of characters in a JTextField
6 * @author Real Gagnon
7 * @authorsWebsite http://tactika.com/realhome/realhome.html
8 * @date 05/07/1999
9 * -----
10 */
11 public class TextFieldLimit extends PlainDocument {
12
13     /**
14      * The maximum number of characters
15      */
16     private int limit;
17
18     /**
19      * Constructs a new TextFieldLimit object
20      * @param limit - the maximum number of characters
21      */
22     public TextFieldLimit(int limit) {
23         super();
24         this.limit = limit;
25     }
26
27     /**
28      * Only inserts a character if the maximum number of characters has not been reached
29      */
30     public void insertString (int offset, String str, AttributeSet attr) throws BadLocationException {
31         if (str == null) return;
32         if ((getLength() + str.length()) <= limit) {
33             super.insertString(offset, str, attr);
34         }
35     }
36 }
```

```
1 import java.util.*;
2
3 /**
4 * -----
5 * Warehouse Application
6 * Constructs the TransactionList object, a list of transactions
7 * @author Alvaro Morales
8 * @date 10/07/2010
9 * @school Markham College
10 * @IDE Eclipse SDK
11 * @computer IBM ThinkPad R52
12 * -----
13 */
14 public class TransactionList {
15
16     /**
17      * The head (first item) of the list of transactions
18      */
19     private TransactionNode head;
20
21     /**
22      * Constructs an empty TransactionList object
23      */
24     public TransactionList(){
25
26     }
27
28     /**
29      * Constructs the TransactionList object, a list of transactions
30      * @param head - the first item of the list
31      */
32     public TransactionList(TransactionNode head) {
33         this.head = head;
34     }
35
36     /**
37
```

```
37     * Gets the head (first item) of the list
38     * @return the head (first item) of the list
39     */
40     public TransactionNode getHead() {
41         return head;
42     }
43
44     /**
45      * Sets the head (first item) of the list
46      * @param head - the head (first item) of the list, of type TransactionNode
47      */
48     public void setHead(TransactionNode head) {
49         this.head = head;
50     }
51
52     /**
53      * Adds a node to the head of the transactions list
54      * @param node - the node to add
55      */
56     public void addToHead(TransactionNode node) {
57         if(head == null){
58             setHead(node);
59         } else {
60             node.setNext(head);
61             head.setPrev(node);
62             setHead(node);
63         }
64     }
65
66     /**
67      * Method searches the list for transactions between specified dates
68      * @param start - the lower boundary of the date search
69      * @param end - the upper boundary of the date search
70      * @return an ArrayList of transactions between the specified dates
71      */
72     public ArrayList searchTransactions(GregorianCalendar start, GregorianCalendar end){
```

```
73     return searchTransactions(head, start, end, new ArrayList());
74 }
75
76 /**
77 * Method is called recursively to search the list for transactions between specified dates
78 * @param current - a pointer to the current node
79 * @param start - the lower boundary of the date search
80 * @param end - the upper boundary of the date search
81 * @param searchResults - an ArrayList of transactions between the specified dates
82 * @return an ArrayList of transactions between the specified dates
83 */
84 private ArrayList searchTransactions(TransactionNode current, GregorianCalendar start,
85                                     GregorianCalendar end, ArrayList searchResults){
86     if(current == null){
87         if(searchResults.isEmpty()){
88             return null;
89         } else {
90             return searchResults;
91         }
92     } else if (current.getTransaction().getDate().compareTo(start) >= 0
93               && current.getTransaction().getDate().compareTo(end) <= 0){
94         searchResults.add(current.getTransaction());
95         return searchTransactions(current.getNext(), start, end, searchResults);
96     } else if (current.getTransaction().getDate().compareTo(end) > 0) {
97         //Stop searching, past the end date
98         searchResults.trimToSize();
99         return searchResults;
100    } else {
101        return searchTransactions(current.getNext(), start, end, searchResults);
102    }
103 }
104
105 /**
106 * Gets a node from the list given the ID of the TransactionRecord stored at that node
107 * @param transactionID - the ID of the TransactionRecord stored at that node
108 * @return the node with that transactionID, or null if not found
```

```
109     */
110    public TransactionNode getNode(int transactionID) {
111        return getNode(getHead(), transactionID);
112    }
113
114    /**
115     * Method is called recursively to get a node from the list given a TransactionRecord ID
116     * @param current - a pointer to the current node
117     * @param transactionID - the ID of the TransactionRecord stored at that node
118     * @return the node with that transactionID, or null if not found
119     */
120    private TransactionNode getNode(TransactionNode current, int transactionID){
121        if(current == null){
122            return null;
123        } else {
124            if(current.getTransaction().getID() == transactionID){
125                return current;
126            } else {
127                return getNode(current.getNext(), transactionID);
128            }
129        }
130    }
131
132    /**
133     * Deletes a node from the Transactions list
134     * @param node - the node to delete
135     */
136    public void deleteTransaction(TransactionNode node){
137        if(node.getPrev() == null){           //it is the head of the list
138            setHead(node.getNext());
139            head.setPrev(null);
140            node.setNext(null);
141            node.setPrev(null);
142        } else if (node.getNext() == null){    //it is the tail of the list
143            node.getPrev().setNext(null);
144            node.setPrev(null);
```

```
145     } else {
146         node.getPrev().setNext(node.getNext());
147         node.getNext().setPrev(node.getPrev());
148         node.setPrev(null);
149         node.setNext(null);
150     }
151 }
152
153 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the TransactionNode object, a node in a doubly linked list
5  * @author Alvaro Morales
6  * @date 10/07/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class TransactionNode {
13
14     /**
15      * The transaction stored in the node
16      */
17     private TransactionRecord transaction;
18
19     /**
20      * The previous transaction node
21      */
22     private TransactionNode prev;
23
24     /**
25      * The next transaction node
26      */
27     private TransactionNode next;
28
29     /**
30      * Constructs the TransactionNode object, a node in a doubly linked list
31      * @param transaction - the transaction stored in the node
32      * @param prev - the previous transaction
33      * @param next - the next transaction
34      */
35     public TransactionNode(TransactionRecord transaction, TransactionNode prev, TransactionNode next) {
36         this.transaction = transaction;
```

```
37         this.prev = prev;
38         this.next = next;
39     }
40
41    /**
42     * Constructs a new Transaction node object
43     * @param transaction - the transaction record stored in this node
44     */
45    public TransactionNode(TransactionRecord transaction){
46        this.transaction = transaction;
47    }
48
49    /**
50     * Gets the next transaction node
51     * @return the next transaction node
52     */
53    public TransactionNode getNext() {
54        return next;
55    }
56
57    /**
58     * Sets the next transaction node
59     * @param next - the next transaction node, of type TransactionNode
60     */
61    public void setNext(TransactionNode next) {
62        this.next = next;
63    }
64
65    /**
66     * Gets the previous transaction node
67     * @return - the previous transaction node
68     */
69    public TransactionNode getPrev() {
70        return prev;
71    }
72}
```

```
73  /**
74   * Sets the previous transaction node
75   * @param prev - the previous transaction node, of type TransactionNode
76   */
77  public void setPrev(TransactionNode prev) {
78      this.prev = prev;
79  }
80
81 /**
82  * Gets the transaction stored in this node
83  * @return the transaction stored in this node
84  */
85  public TransactionRecord getTransaction() {
86      return transaction;
87  }
88
89 /**
90  * Sets the transaction stored in this node
91  * @param transaction - the transaction stored in this node, of type TransactionRecord
92  */
93  public void setTransaction(TransactionRecord transaction) {
94      this.transaction = transaction;
95  }
96
97 }
```

```
1 import javax.swing.table.AbstractTableModel;
2 import java.util.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the New User screen and handles new user creation
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15
16 public class TransactionProcessingTableModel extends AbstractTableModel {
17
18     /**
19      * An ArrayList of items to process (either enter into or withdraw from the warehouse)
20      */
21     private ArrayList itemsToProcess;
22
23     /**
24      * An array of transactions
25      */
26     private TransactionRecord[] transactions;
27
28     /**
29      * Constructs a SearchResultsTableModel
30      * @param itemsToProcess - an ArrayList of the search results
31      */
32     public TransactionProcessingTableModel(ArrayList itemsToProcess) {
33         this.itemsToProcess = itemsToProcess;
34
35         if(itemsToProcess != null){
36             transactions = new TransactionRecord[itemsToProcess.size()];
```

```
37     //Initialize the transactions
38     for(int i=0;i<transactions.length;i++){
39         transactions[i] = new TransactionRecord();
40     }
41 }
42 }
43 }
44 }
45
46 /**
47 * Returns the number of columns in the table
48 * @return - the number of columns in the table
49 */
50 public int getColumnCount(){
51     return 5;
52 }
53
54 /**
55 * Returns the number of rows in the table
56 * @return - the number of rows in the table (the number of items in the ArrayList)
57 */
58 public int getRowCount(){
59     if(itemsToProcess!= null){
60         return itemsToProcess.size();
61     } else {
62         return 0;
63     }
64 }
65 }
66
67 /**
68 * Returns the object in a cell of the table at a given row-column location
69 * @param row - row number
70 * @param col - column number
71 * @return - the object at that row-column location
72 */
```

```
73 public Object getValueAt(int row, int col){  
74     Item item = (Item)itemsToProcess.get(row);  
75     switch(col){  
76         case 0:  
77             return item.getCode();  
78         case 1:  
79             return item.getName();  
80         case 2:  
81             return item.getUm().toString();  
82         case 3:  
83             return new Integer(Transactions.getQuantityInStock(item));  
84         case 4:  
85             if(transactions[row].getQuantity() != 0){  
86                 return new Integer(transactions[row].getQuantity());  
87             } else {  
88                 return new Integer(0);  
89             }  
90         default:  
91             return null;  
92     }  
93 }  
94 /**  
95 * Returns the name of a column  
96 * @param col - the column number  
97 * @return the name of a column  
98 */  
99 public String getColumnName(int col){  
100     switch(col){  
101         case 0:  
102             return "Code";  
103     }  
104 }
```

```
109     case 1:
110         return "Name";
111     case 2:
112         return "U.M.";
113     case 3:
114         return "In Stock";
115     case 4:
116         return "Quantity*";
117     default:
118         return null;
119     }
120 }
121 /**
122 * Returns the class of a column
123 * @param col - the column number
124 * @return the Class of a column
125 */
126 public Class getColumnClass(int col){
127
128     switch(col){
129
130         case 0:
131             return String.class;
132         case 1:
133             return String.class;
134         case 2:
135             return String.class;
136         case 3:
137             return Integer.class;
138         case 4:
139             return Integer.class;
140         default:
141             return null;
142     }
143 }
144 }
```

```
145
146     }
147
148     /**
149      * Returns if a cell is editable (only the 'Quantity' column)
150      * @return false unless the col index is column 4 (quantity)
151      */
152     public boolean isCellEditable(int row, int col){
153         if(col == 4){
154             return true;
155         } else {
156             return false;
157         }
158     }
159
160     /**
161      * Returns an ArrayList of the search results
162      * @return an ArrayList of the search results
163      */
164     public ArrayList getSearchResults(){
165         return itemsToProcess;
166     }
167
168     /**
169      * Sets the value for the quantity of the transaction
170      */
171     public void setValueAt(Object field, int row, int col){
172
173         switch(col){
174             case 4:
175                 transactions[row].setQuantity(((Integer)field).intValue());
176                 break;
177             default:
178                 break;
179         }
180     }
```

```
181
182     /**
183      * Gets the transactions
184      * @return the transactions to process
185      */
186     public TransactionRecord[] getTransactions() {
187         return transactions;
188     }
189
190 }
```

```
1 import java.util.*;
2 import java.io.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the TransactionRecord object that has information regarding an item transaction
8 * @author Alvaro Morales
9 * @date 10/07/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15 public class TransactionRecord {
16
17     /**
18      * The transaction's unique identifier code
19      */
20     private int ID;
21
22     /**
23      * The type of transaction (Item entry or exit)
24      */
25     private int type;
26
27     /**
28      * The date when the transaction happened
29      */
30     private GregorianCalendar date;
31
32     /**
33      * Amount of the item's unit of measurement to be transacted (entered or removed)
34      */
35     private int quantity;
36 }
```

```
37  /**
38   * The amount remaining of the item's unit of measurement after this transaction
39   */
40  private int balance;
41
42  /**
43   * The username of the worker who processed the transaction
44   */
45  private String warehouseWorker;
46
47  /**
48   * The item that is being transacted
49   */
50  private Item item;
51
52  /**
53   * The document noting the transaction
54   */
55  private String document;
56
57  /**
58   * The person/company delivering or withdrawing the item
59   */
60  private String thirdParty;
61
62  /**
63   * Constructs a TransactionRecord object that has information regarding an item transaction
64   * @param ID - the transaction's unique identifier code
65   * @param date - the date when the transaction happened
66   * @param quantity - amount of the item's unit of measurement to be transacted (entered or removed)
67   * @param balance - the amount remaining of the item's unit of measurement after this transaction
68   * @param warehouseWorker - the username of the worker who processed the transaction
69   * @param item - the item that is being transacted
70   * @param document - the document noting the transaction
71   * @param thirdParty - the person/company delivering or withdrawing the item
72   */
```

```
73     public TransactionRecord(int ID, int type, GregorianCalendar date, int quantity, int balance,
74         String warehouseWorker, String document, String thirdParty, Item item) {
75         this.ID = ID;
76         this.type = type;
77         this.date = date;
78         this.quantity = quantity;
79         this.balance = balance;
80         this.warehouseWorker = warehouseWorker;
81         this.document = document;
82         this.thirdParty = thirdParty;
83         this.item = item;
84     }
85
86     /**
87      * Constructs an empty TransactionRecord object
88      */
89     public TransactionRecord(){
90
91     }
92
93     /**
94      * Gets the transaction balance
95      * @return the transaction balance
96      */
97     public int getBalance() {
98         return balance;
99     }
100
101    /**
102     * Sets the transaction balance
103     * @param balance - the transaction balance, of type int
104     */
105    public void setBalance(int balance) {
106        this.balance = balance;
107    }
108}
```

```
109     /**
110      * Gets the date of the transaction
111      * @return the date of the transaction
112      */
113     public GregorianCalendar getDate() {
114         return date;
115     }
116
117     /**
118      * Sets the date of the transaction
119      * @param date - the date of the transaction, of type Calendar
120      */
121     public void setDate(GregorianCalendar date) {
122         this.date = date;
123     }
124
125     /**
126      * Gets the ID of the transaction
127      * @return - the ID of the transaction
128      */
129     public int getID() {
130         return ID;
131     }
132
133     /**
134      * Sets the ID of the transaction
135      * @param ID - the ID of the transaction, of type int
136      */
137     public void setID(int ID) {
138         this.ID = ID;
139     }
140
141     /**
142      * Gets the item being transacted
143      * @return - the item being transacted
144      */
```

```
145     public Item getItem() {
146         return item;
147     }
148
149     /**
150      * Sets the item being transacted
151      * @param item - the item being transacted, of type Item
152      */
153     public void setItem(Item item) {
154         this.item = item;
155     }
156
157     /**
158      * Gets the quantity of the transaction
159      * @return - the quantity of the transaction
160      */
161     public int getQuantity() {
162         return quantity;
163     }
164
165     /**
166      * Sets the quantity of the transaction
167      * @param quantity - the quantity of the transaction, of type int
168      */
169     public void setQuantity(int quantity) {
170         this.quantity = quantity;
171     }
172
173     /**
174      * Gets the username of the warehouse that processed the transaction
175      * @return the username of the warehouse that processed the transaction
176      */
177     public String getWarehouseWorker() {
178         return warehouseWorker;
179     }
180 }
```

```
181  /**
182   * Sets the username of the warehouse that processed the transaction
183   * @param warehouseWorker - the username of the warehouse that processed the transaction, of type String
184   */
185  public void setWarehouseWorker(String warehouseWorker) {
186      this.warehouseWorker = warehouseWorker;
187  }
188
189 /**
190  * Gets the Transaction's document (Proof of reception or Exit voucher)
191  * @return the transaction's document
192  */
193  public String getDocument() {
194      return document;
195  }
196
197 /**
198  * Sets the transaction's document
199  * @param document - the transaction's document, of type String
200  */
201  public void setDocument(String document) {
202      this.document = document;
203  }
204
205 /**
206  * Gets the third party that requested/delivered the transaction
207  * @return the third party
208  */
209  public String getThirdParty() {
210      return thirdParty;
211  }
212
213 /**
214  * Sets the third party that requested/delivered the transaction
215  * @param thirdParty - the third party, of type String
216  */
```

```
217     public void setThirdParty(String thirdParty) {
218         this.thirdParty = thirdParty;
219     }
220
221     /**
222      * Gets the type of transaction
223      * @return the type of transaction
224      */
225     public int getType() {
226         return type;
227     }
228
229     /**
230      * Sets the type of transaction
231      * @param type - the type of transaction, of type int
232      */
233     public void setType(int type) {
234         this.type = type;
235     }
236
237     /**
238      * Processes a transaction (adds it to the Transactions file)
239      * @param transaction - the transaction to process
240      */
241     public void processTransaction(){
242         try{
243             int itemID = getItem().getID();
244             File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + itemID + ".txt");
245             File tmpFile = new File(ApplicationConstants.TRANSACTIONS_FOLDER + itemID + "_tmp.txt");
246             FileReader reader = new FileReader(file);
247             BufferedReader buff = new BufferedReader(reader);
248             FileWriter writer = new FileWriter(tmpFile);
249             BufferedWriter buffwriter = new BufferedWriter(writer);
250
251             boolean eof = false;           //stores if the end of the file (eof) has been reached
252             int transactionID = 0;        //will record the ID of this transaction
```

```
253 TransactionRecord previousTransaction = null;
254
255 while(!eof){
256     String line = buff.readLine();
257     if(line == null){
258         eof = true;           //the end of the file has been reached
259     } else {
260         //all transactions in this file will be of the same item
261         previousTransaction = Transactions.readTransaction(getItem(), line);
262         buffwriter.write(line);
263         buffwriter.newLine();
264         transactionID++;
265     }
266 }
267
268 if(previousTransaction == null){
269     setBalance(getQuantity());           //the first transaction will always be an Item Entry
270 } else {
271     if(getType() == ApplicationConstants.ITEM_ENTRY){
272         setBalance(previousTransaction.getBalance() + getQuantity());
273     } else {
274         setBalance(previousTransaction.getBalance() - getQuantity());
275     }
276 }
277
278 setID(transactionID);
279 buffwriter.write(toString());
280 buffwriter.newLine();
281
282 buffwriter.close();
283 buff.close();
284
285 //Rename temporary file to old file
286 file.delete();
287 tmpFile.renameTo(file);
288
```

```
289     } catch (Exception e){
290         JOptionPane.showMessageDialog(
291             MainScreen.contentPane, "One or files cannot be accessed. \n"
292             + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
293     }
294 }
295 }
296 /**
297 * Outputs a TransactionRecord object to a tokenized String formatted for the transactions file
298 */
299 public String toString(){
300     int day = getDate().get(Calendar.DAY_OF_MONTH);
301     int month = getDate().get(Calendar.MONTH);
302     int year = getDate().get(Calendar.YEAR);
303
304     return (getID() + "|" + getType() + "|" + day + "|" + month + "|" + year + "|" +
305             getQuantity() + "|" + getBalance() + "|" + getWarehouseWorker() + "|"
306             + getDocument() + "|" + getThirdParty());
307 }
308 }
309 }
310 }
```

```
1 import java.io.*;
2 import java.util.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Class contains methods relevant to Transaction processing
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15 public class Transactions {
16
17     /**
18      * Gets the quantity in stock of an item
19      * @param item - the item
20      * @return the quantity in stock of that item
21      */
22     public static int getQuantityInStock(Item item){
23         try {
24             int ID = item.getID();
25
26             File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + ID + ".txt");
27             FileReader reader = new FileReader(file);
28             BufferedReader buff = new BufferedReader(reader);
29
30             boolean eof = false;           //stores if the end of the file (eof) has been reached
31             TransactionRecord transaction = null;
32
33             while(!eof){
34                 String line = buff.readLine();
35                 if(line == null){
36                     eof = true;           //the end of the file has been reached
37             }
38         }
39     }
40 }
```

```
37         } else {
38             transaction = readTransaction(item, line);
39         }
40     }
41
42     buff.close();
43     reader.close();
44
45     return transaction.getBalance();
46 } catch (Exception e){
47     return 0;
48 }
49
50 }
51
52 /**
53 * Creates a TransactionRecord object from a tokenized String input
54 * @param line - a line read from a Transactions file
55 * @return - a TransactionRecord object
56 */
57 public static TransactionRecord readTransaction(Item item, String line){
58     StringTokenizer tokenizer = new StringTokenizer(line, "|");           //Tokenize string
59
60     if (tokenizer.countTokens() == 10){          //preliminary error checking
61         int ID = new Integer(tokenizer.nextToken()).intValue();
62         int type = new Integer(tokenizer.nextToken()).intValue();
63         int day = new Integer(tokenizer.nextToken()).intValue();
64         int month = new Integer(tokenizer.nextToken()).intValue();
65         int year = new Integer(tokenizer.nextToken()).intValue();
66         int quantity = new Integer(tokenizer.nextToken()).intValue();
67         int balance = new Integer(tokenizer.nextToken()).intValue();
68         String warehouseWorker = tokenizer.nextToken();
69         String document = tokenizer.nextToken();
70         String thirdParty = tokenizer.nextToken();
71
72         GregorianCalendar date = new GregorianCalendar(year, month, day);
```

```
73
74     return new TransactionRecord(ID, type, date, quantity, balance, warehouseWorker,
75         document, thirdParty, item);
76     }
77     return null;
78 }
79
80 }
```

```
1 import java.util.*;
2 import javax.swing.table.AbstractTableModel;
3
4 /**
5 * -----
6 * Warehouse Application
7 * A TableModel for the displaying of transactions
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15 public class TransactionsTableModel extends AbstractTableModel {
16
17     /**
18      * An ArrayList of TransactionRecord objects that will be displayed in the table
19      */
20     private ArrayList transactionsToDisplay;
21
22     /**
23      * Constructs a TransactionsTableModel object to display transactions in a JTable
24      * @param transactionsToDisplay - an ArrayList of TransactionRecord objects that will be displayed in the table
25      */
26     public TransactionsTableModel(ArrayList transactionsToDisplay) {
27         this.transactionsToDisplay = transactionsToDisplay;
28     }
29
30     /**
31      * Returns the number of columns in the table
32      * @return - the number of columns in the table
33      */
34     public int getColumnCount() {
35         return 8;
36     }
```

```
37
38 /**
39  * Returns the number of rows in the table
40  * @return - the number of rows in the table (the number of items in the ArrayList)
41  */
42 public int getRowCount(){
43     if(transactionsToDisplay!= null){
44         return transactionsToDisplay.size();
45     } else {
46         return 0;
47     }
48 }
49
50 /**
51  * Returns the object in a cell of the table at a given row-column location
52  * @param row - row number
53  * @param col - column number
54  * @return - the object at that row-column location
55  */
56 public Object getValueAt(int row, int col){
57
58     TransactionRecord transaction = (TransactionRecord)transactionsToDisplay.get(row);
59
60     switch(col){
61         case 0:
62             GregorianCalendar transactionDate = transaction.getDate();
63             int day = transactionDate.get(Calendar.DAY_OF_MONTH);
64             int month = transactionDate.get(Calendar.MONTH);
65             int year = transactionDate.get(Calendar.YEAR);
66             return (day + "/" + month + "/" + year);
67         case 1:
68             if(transaction.getType() == ApplicationConstants.ITEM_ENTRY){
69                 return "Entry";
70             } else {
71                 return "Exit";
72             }
73     }
74 }
```

```
73     case 2:
74         return new Integer(transaction.getQuantity());
75     case 3:
76         return new Integer(transaction.getBalance());
77     case 4:
78         return transaction.getDocument();
79     case 5:
80         if(transaction.getType() == ApplicationConstants.ITEM_ENTRY) {
81             return transaction.getThirdParty();
82         } else {
83             return null;
84         }
85     case 6:
86         if(transaction.getType() == ApplicationConstants.ITEM_EXIT) {
87             return transaction.getThirdParty();
88         } else {
89             return null;
90         }
91     case 7:
92         return transaction.getWarehouseWorker();
93     default:
94         return null;
95     }
96 }
97 /**
98 * Returns the name of a column
99 * @param col - the column number
100 * @return the name of a column
101 */
102 public String getColumnName(int col){
103
104     switch(col){
105
106         case 0:
107             return "Date";
108     }
109 }
```

```
109     case 1:  
110         return "Type";  
111     case 2:  
112         return "Quantity";  
113     case 3:  
114         return "Balance";  
115     case 4:  
116         return "Document";  
117     case 5:  
118         return "Delivered By";  
119     case 6:  
120         return "Requested By";  
121     case 7:  
122         return "Warehouse Staff";  
123     default:  
124         return null;  
125     }  
126 }  
127  
128 /**
129 * Returns if a cell is editable
130 * @return a boolean indicating if the cell is editable
131 */
132 public boolean isCellEditable(int row, int col){
133     return false;
134 }
135  
136 /**
137 * Gets the ArrayList of TransactionRecord objects to display
138 * @return the ArrayList of TransactionRecord objects to display
139 */
140 public ArrayList getTransactionsToDisplay() {
141     return transactionsToDisplay;
142 }
143
144 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the UM (Unit of Measurement) object that has information regarding unit of measurements for items
5  * @author Alvaro Morales
6  * @date 24/06/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class UM {
13
14     /**
15      * The UM's code
16      */
17     private short code;
18
19     /**
20      * The UM's name
21      */
22     private String name;
23
24     /**
25      * Constructs a new UM (Unit of Measurement) object
26      * @param code - the UM's code of type short
27      * @param name - the UM's name of type String
28      */
29     public UM(short code, String name){
30         this.code = code;
31         this.name = name;
32     }
33
34     /**
35      * Gets the UM's (Unit of Measurement's) code
36      * @return the UM's code
```

```
37     */
38     public short getCode() {
39         return code;
40     }
41
42     /**
43      * Sets the UM's (Unit of Measurement's) code
44      * @param code - the UM's code of type short
45      */
46     public void setCode(short code) {
47         this.code = code;
48     }
49
50     /**
51      * Gets the UM's (Unit of Measurement's) name
52      * @return the UM's name
53      */
54     public String getName() {
55         return name;
56     }
57
58     /**
59      * Sets the UM's (Unit of Measurement's) name
60      * @param name - the UM's name of type String
61      */
62     public void setName(String name) {
63         this.name = name;
64     }
65
66     /**
67      * Returns the name of the Unit of Measurement (UM)
68      */
69     public String toString(){
70         return getName();
71     }
72 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Constructs the User object that has information regarding the user of the application
5  * @author Alvaro Morales
6  * @date 03/06/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class User {
13
14     /**
15      * The user's unique identifier name
16      */
17     private String username;
18
19     /**
20      * The user's password
21      */
22     private String password;
23
24     /**
25      * The user's full name
26      */
27     private String fullName;
28
29     /**
30      * Stores whether the user has administrator permissions
31      */
32     private boolean isAdmin;
33
34     /**
35      * Stores whether the user has item entry permissions
36      */
```

```
37  private boolean isEntry;
38
39  /**
40   * Stores whether the user has item exit permissions
41   */
42  private boolean isExit;
43
44  /**
45   * Stores whether the user is allowed to login
46   */
47  private boolean isEnabled;
48
49  /**
50   * Constructs a User object
51   * @param username - the unique identifier name for a user
52   * @param password - the password to have access to the application
53   * @param fullName - the user's full name
54   * @param isAdmin - indicates if the user has administrator permissions
55   * @param isEntry - indicates if the user has permission to execute item entry related operations
56   * @param isExit - indicates if the user has permission to execute item exit related operations
57   * @param isEnabled - indicates if the user is enabled to access the system
58   */
59  public User(String username, String password, String fullName, boolean isAdmin, boolean isEntry,
60             boolean isExit, boolean isEnabled){
61      this.username = username;
62      this.password = password;
63      this.fullName = fullName;
64      this.isAdmin = isAdmin;
65      this.isEntry = isEntry;
66      this.isExit = isExit;
67      this.isEnabled = isEnabled;
68  }
69
70 /**
71  * Returns the user's full name
72  * @return the user's full name
```

```
73     */
74     public String getFullName() {
75         return fullName;
76     }
77
78     /**
79      * Sets the user's full name
80      * @param fullName - the user's full name
81      */
82     public void setFullName(String fullName) {
83         this.fullName = fullName;
84     }
85
86     /**
87      * Returns the boolean that stores whether the user has admin permissions or not
88      * @return true if the user has admin permissions, else false
89      */
90     public boolean isAdmin() {
91         return isAdmin;
92     }
93
94     /**
95      * Sets the admin permissions of the user
96      * @param isAdmin - true if the user has admin permissions, else false
97      */
98     public void setAdmin(boolean isAdmin) {
99         this.isAdmin = isAdmin;
100    }
101
102    /**
103     * Returns the boolean that stores whether the user is enabled or not
104     * @return true if the user is enabled, else false
105     */
106    public boolean isEnabled() {
107        return isEnabled;
108    }
```

```
109
110     /**
111      * Sets the boolean that stores whether the user is enabled or not
112      * @param isEnabled - true if the user is enabled, else false
113      */
114     public void setEnabled(boolean isEnabled) {
115         this.isEnabled = isEnabled;
116     }
117
118     /**
119      * Returns the boolean that stores whether the user has entry permissions or not
120      * @return true if the user has entry permissions, else false
121      */
122     public boolean isEntry() {
123         return isEntry;
124     }
125
126     /**
127      * Sets the boolean that stores whether the user has entry permissions or not
128      * @param isEntry - true if the user has entry permissions, else false
129      */
130     public void setEntry(boolean isEntry) {
131         this.isEntry = isEntry;
132     }
133
134     /**
135      * Returns the boolean that stores whether the user has exit permissions or not
136      * @return true if the user has exit permissions, else false
137      */
138     public boolean isExit() {
139         return isExit;
140     }
141
142     /**
143      * Sets the boolean that stores whether the user has exit permissions or not
144      * @param isExit
```

```
145     */
146     public void setExit(boolean isExit) {
147         this.isExit = isExit;
148     }
149
150     /**
151      * Gets the user's password
152      * @return the user's password
153      */
154     public String getPassword() {
155         return password;
156     }
157
158     /**
159      * Sets the password of the user, that should not be longer than 30 characters
160      * @param password - the user's password
161      * @throws Exception if password is longer than 30 characters
162      */
163     public void setPassword(String password) throws Exception {
164
165         if(password.length()<=30){
166             this.password = password;
167         } else{
168             throw new Exception();
169         }
170     }
171
172     /**
173      * Gets the user's username
174      * @return the user's username
175      */
176     public String getUsername() {
177         return username;
178     }
179
180 }
```

```
181  /**
182   * Sets the username of the user, that should not be longer than 20 characters
183   * @param username - the user's unique identifier name
184   * @throws Exception if username is longer than 20 characters
185   */
186  public void setUsername(String username) throws Exception {
187      if(username.length()<=30){
188          this.username = username;
189      } else{
190          throw new Exception();
191      }
192  }
193
194  /**
195   * Returns an int from a boolean
196   * @param b - the boolean
197   * @return 1 if true, else 0
198   */
199  public int booleanToInt(boolean b){
200      return (b) ? 1:0;
201  }
202
203  /**
204   * Outputs a User object in tokenized form with '|' as the token, for use with the Users file
205   * @return the tokenized user object
206   */
207  public String toString(){
208      return (getUsername() + "|" + getPassword() + "|"
209          + getFullName() + "|"
210          + booleanToInt(isAdmin()) + "|"
211          + booleanToInt(isEntry()) + "|"
212          + booleanToInt(isExit()) + "|"
213          + booleanToInt(isEnabled())+ "|");
214  }
215
216 }
```

```
1 import javax.swing.table.AbstractTableModel;
2 import javax.swing.JButton;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the New User screen and handles new user creation
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15
16 public class UsersTableModel extends AbstractTableModel {
17
18     /**
19      * An array of users
20      */
21     private User[] users;
22
23     /**
24      * Constructs a UsersTableModel
25      * @param users - array of all users
26      */
27     public UsersTableModel(User[] users) {
28         this.users = users;
29     }
30
31     /**
32      * Returns the number of columns in the table
33      * @return - the number of columns in the table (7)
34      */
35     public int getColumnCount() {
36         return 7;
```

```
37     }
38
39     /**
40      * Returns the number of rows in the table
41      * @return - the number of rows in the table (the number of users in the file)
42      */
43     public int getRowCount(){
44         return users.length;
45     }
46
47     /**
48      * Returns the object in a cell of the table at a given row-column location
49      * @param row - row number
50      * @param col - column number
51      * @return - the object at that row-column location
52      */
53     public Object getValueAt(int row, int col){
54
55         User user = users[row];
56
57         switch(col){
58             case 0:
59                 return user.getUsername();
60             case 1:
61                 return user.getPassword();
62             case 2:
63                 return user.getFullName();
64             case 3:
65                 return new Boolean(user.isAdmin());
66             case 4:
67                 return new Boolean(user.isEntry());
68             case 5:
69                 return new Boolean(user.isExit());
70             case 6:
71                 return new Boolean(user.isEnabled());
72             case 7:
```

```
73         return new JButton("Edit");
74     default:
75         return null;
76     }
77 }
78 /**
79 * Returns the name of a column
80 * @param col - the column number
81 * @return the name of a column
82 */
83 public String getColumnName(int col){
84
85     switch(col){
86
87     case 0:
88         return "Username";
89     case 1:
90         return "Password";
91     case 2:
92         return "Full Name";
93     case 3:
94         return "Admin";
95     case 4:
96         return "Entry";
97     case 5:
98         return "Exit";
99     case 6:
100        return "Enabled";
101    default:
102        return null;
103    }
104 }
105 /**
106 */
107 /**
108 */
```

```
109     * Returns the class of a column
110     * @param col - the column number
111     * @return the Class of a column
112     */
113    public Class getColumnClass(int col){
114
115        switch(col){
116
117            case 0:
118                return String.class;
119            case 1:
120                return String.class;
121            case 2:
122                return String.class;
123            case 3:
124                return Boolean.class;
125            case 4:
126                return Boolean.class;
127            case 5:
128                return Boolean.class;
129            case 6:
130                return Boolean.class;
131            default:
132                return null;
133        }
134    }
135
136
137    /**
138     * Returns if a cell is editable
139     * @return false, regardless of row-column index
140     */
141    public boolean isCellEditable(int row, int col){
142        return false;
143    }
144}
```

```
145     /**
146      * Returns all the users in the table
147      * @return the user array
148      */
149     public User[] getAllUsers() {
150         return users;
151     }
152 }
153 }
```

```
1  /**
2  * -----
3  * Warehouse Application
4  * Stores the number of aisles in the warehouse and the number of rows and columns per rack
5  * @author Alvaro Morales
6  * @date 03/07/2010
7  * @school Markham College
8  * @IDE Eclipse SDK
9  * @computer IBM ThinkPad R52
10 *
11 */
12 public class WarehouseLocations {
13
14     /**
15      * An array of aisles in the warehouse
16      */
17     private Integer[] aisles = completeIntegerArray(70);
18
19     /**
20      * An array of rows in a rack in the warehouse
21      */
22     private String[] rows = completeStringArray('E');
23
24     /**
25      * An array of columns in a rack in the warehouse
26      */
27     private Integer[] columns = completeIntegerArray(10);
28
29     /**
30      * Constructs a new WarehouseLocations object.
31      * 70 aisles from 1 - 70,
32      * 5 rows per rack from A - E,
33      * 10 columns per rack from 1 - 10.
34      */
35     public WarehouseLocations() {
36
```

```
37     }
38
39     /**
40      * Completes an int array given its last value
41      * @param intArray - the int array to complete
42      * @param lastAisle - the last value
43      */
44     private Integer[] completeIntegerArray(int lastValue){
45         Integer[] intArray = new Integer[lastValue];
46
47         int value = 0;
48
49         for (int i=0;i<intArray.length;i++){
50             intArray[i] = new Integer(value+1);
51             value = value + 1;
52         }
53
54         return intArray;
55     }
56
57     /**
58      * Complete a char array given its last uppercase char from the first value 'A'
59      * @param charArray - the char array to complete
60      * @param lastChar - the last uppercase char
61      */
62     private String[] completeStringArray(char lastChar){
63         char firstChar = (char)((int)('A')-1);
64         int size = (int)lastChar - (int)firstChar;
65
66         String[] stringArray = new String[size];
67
68         char character = (int)('A')-1;
69
70         for (int i=0;i<stringArray.length;i++){
71             stringArray[i] = new String(" " + (char)(character+1));
72             character = (char)(character+1);
```

```
73
74         }
75     return stringArray;
76 }
77
78 /**
79  * Gets the array of aisle numbers
80  * @return the array of aisle numbers
81  */
82 public Integer[] getAisles() {
83     return aisles;
84 }
85
86 /**
87  * Gets the array of column numbers
88  * @return the array of column numbers
89  */
90 public Integer[] getColumns() {
91     return columns;
92 }
93
94 /**
95  * Gets the array of row characters
96  * @return the array of row characters
97  */
98 public String[] getRows() {
99     return rows;
100 }
101
102 }
```

```
1 import javax.swing.*;
2 import java.awt.BorderLayout;
3 import java.awt.event.*;
4
5 /**
6 * -----
7 * Warehouse Application
8 * Constructs the WarehouseMenuBar object
9 * @author Alvaro Morales
10 * @date 13/06/2010
11 * @school Markham College
12 * @IDE Eclipse SDK
13 * @computer IBM ThinkPad R52
14 * -----
15 */
16 public class WarehouseMenuBar extends javax.swing.JMenuBar {
17
18     /**
19      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
20      */
21
22     /**
23      * GUI components generated by Jigloo
24      */
25
26     private JMenuItem mnuManageUsers;
27     private JSeparator spFileMenu;
28     private JMenuItem mnuAccountSettings;
29     private JMenuItem mnuAbout;
30     private JMenu mnuHelp;
31     private JMenu mnuAdmin;
32     private JMenuItem mnuCreateNewItem;
33     private JMenuItem mnuExit;
34     private JMenuItem mnuReturn;
35     private JMenuItem mnuItemExit;
36     private JMenuItem mnuItemEntry;
```

```
37     private JMenu mnuStockMaintenance;
38     private JMenuItem mnuItemSearch;
39     private JMenu mnuItemRecords;
40     private JMenu mnuFile;
41
42     /**
43      * The user that logged in
44      */
45     private User user;
46
47     /**
48      * Constructs a WarehouseMenuBar object
49      * @param user - the user that logged in
50      * @param main - the MainScreen object that holds the menu bar
51      */
52     public WarehouseMenuBar(User user, MainScreen main){
53
54         final MainScreen MAIN_SCREEN = main;
55         this.user = user;
56
57         {
58             mnuFile = new JMenu();
59             this.add(mnuFile);
60             mnuFile.setText("File");
61             {
62                 mnuAccountSettings = new JMenuItem();
63                 mnuFile.add(mnuAccountSettings);
64                 mnuAccountSettings.setText("Account Settings");
65                 mnuAccountSettings.addActionListener(new ActionListener() {
66                     public void actionPerformed (ActionEvent e){
67                         openAccountSettings();
68                     }
69                 });
70             }
71             {
72                 spFileMenu = new JSeparator();
```

```
73         mnuFile.add(spFileMenu);
74     }
75     {
76         mnuReturn = new JMenuItem();
77         mnuFile.add(mnuReturn);
78         mnuReturn.setText("Return to Main Menu");
79         mnuReturn.addActionListener(new ActionListener(){
80             public void actionPerformed(ActionEvent e){
81                 ChangeScreen.setBlankScreen(getUser());
82             }
83         });
84     }
85     {
86         mnuExit = new JMenuItem();
87         mnuFile.add(mnuExit);
88         mnuExit.setText("Exit");
89         mnuExit.addActionListener(new ActionListener(){
90             public void actionPerformed(ActionEvent e){
91                 System.exit(0);
92             }
93         });
94     }
95 }
96 {
97     mnuItemRecords = new JMenu();
98     this.add(mnuItemRecords);
99     mnuItemRecords.setText("Item Records");
100 {
101     //Construct items only relevant to user permissions
102     if(user.isEntry() || user.isAdmin()){
103         mnuCreateNewItem = new JMenuItem();
104         mnuItemRecords.add(mnuCreateNewItem);
105         mnuCreateNewItem.setText("New Item");
106         mnuCreateNewItem.setIcon(new ImageIcon(getClass().getClassLoader().getResource(
107             ApplicationConstants.CREATE_ITEM_ICON)));
108         mnuCreateNewItem.addActionListener(new ActionListener(){
```

```
109     public void actionPerformed(ActionEvent e){
110         ChangeScreen.setCreateNewItemScreen(getUser());
111     }
112 }
113 }
114 }
115 {
116 {
117     mnuItemSearch = new JMenuItem();
118     mnuItemRecords.add(mnuItemSearch);
119     mnuItemSearch.setText("Item Search");
120     mnuItemSearch.setIcon(new ImageIcon(getClass().getClassLoader().getResource(
121         ApplicationConstants.ITEM_ENQUIRY_ICON)));
122     mnuItemSearch.addActionListener(new ActionListener(){
123         public void actionPerformed(ActionEvent e){
124             ChangeScreen.setItemSearchScreen(getUser());
125         }
126     });
127 }
128 }
129 {
130     mnuStockMaintenance = new JMenu();
131     this.add(mnuStockMaintenance);
132     mnuStockMaintenance.setText("Stock Maintenance");
133 {
134     //Construct items only relevant to user permissions
135     if(user.isEntry() || user.isAdmin()){
136         mnuItemEntry = new JMenuItem();
137         mnuStockMaintenance.add(mnuItemEntry);
138         mnuItemEntry.setText("Stock Entry");
139         mnuItemEntry.setIcon(new ImageIcon(getClass().getClassLoader().getResource(
140             ApplicationConstants.ITEM_ENTRY_ICON)));
141         mnuItemEntry.addActionListener(new ActionListener(){
142             public void actionPerformed(ActionEvent e){
143                 ChangeScreen.setMultipleItemSearchScreen(
144                     getUser(), ApplicationConstants.ITEM_ENTRY);
```

```
145             }
146         });
147     }
148
149     {
150         /*
151          //Construct items only relevant to user permissions
152          if(user.isExit() || user.isAdmin()){
153              mnuItemExit = new JMenuItem();
154              mnuStockMaintenance.add(mnuItemExit);
155              mnuItemExit.setText("Item Exit");
156              mnuItemExit.setIcon(new ImageIcon(getClass().getClassLoader().getResource(
157                  ApplicationConstants.ITEM_EXIT_ICON)));
158              mnuItemExit.addActionListener(new ActionListener(){
159                  public void actionPerformed(ActionEvent e){
160                      ChangeScreen.setMultipleItemSearchScreen(
161                          getUser(), ApplicationConstants.ITEM_EXIT);
162                  }
163              });
164          }
165      }
166  }
167
168  {
169      /*
170      if(user.isAdmin()){
171          mnuAdmin = new JMenu();
172          this.add(mnuAdmin);
173          mnuAdmin.setText("Admin");
174          {
175              mnuManageUsers = new JMenuItem();
176              mnuAdmin.add(mnuManageUsers);
177              mnuManageUsers.setText("Manage Users");
178              mnuManageUsers.addActionListener(new ActionListener(){
179                  public void actionPerformed(ActionEvent e){
180                      ChangeScreen.setManageUsersScreen(getUser());
```

```
181             }
182         });
183     }
184 }
185
186 {
187     mnuHelp = new JMenu();
188     this.add(mnuHelp);
189     mnuHelp.setText("Help");
190     {
191         mnuAbout = new JMenuItem();
192         mnuHelp.add(mnuAbout);
193         mnuAbout.setText("About");
194         mnuAbout.addActionListener(new ActionListener() {
195             public void actionPerformed (ActionEvent e){
196                 displayAboutMessage();
197             }
198         });
199     }
200 }
201
202 }
203
204 /**
205 * End of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
206 */
207
208 /**
209 * Gets the user that logged in
210 * @return the user that logged in
211 */
212
213 public User getUser(){
214     return user;
215 }
216
```

```
217     /**
218      * Opens the AccountSettings screen
219      */
220     public void openAccountSettings(){
221         AccountSettings settings = new AccountSettings(user);
222         settings.setVisible(true);
223         settings.setLocation(60, 180);
224     }
225
226     /**
227      * Displays the application information
228      */
229     public void displayAboutMessage(){
230         JOptionPane.showMessageDialog(MainScreen.contentPane, "Developed by Alvaro Morales "
231             + "for Cementos Lima S.A.\n ©2010. All Rights Reserved", "Warehouse Application",
232             JOptionPane.INFORMATION_MESSAGE);
233     }
234
235 }
```

```
1 import java.awt.event.*;
2 import javax.swing.*;
3
4 /**
5 * -----
6 * Warehouse Application
7 * Constructs the application's toolbar
8 * @author Alvaro Morales
9 * @date 10/06/2010
10 * @school Markham College
11 * @IDE Eclipse SDK
12 * @computer IBM ThinkPad R52
13 * -----
14 */
15
16 public class WarehouseToolbar extends javax.swing.JToolBar {
17
18     /**
19      * Start of the code generated using CloudGarden's Jigloo SWT/Swing GUI Builder
20      */
21
22     /**
23      * GUI components generated by Jigloo
24      */
25     private JButton btnSearch;
26     private JButton btnExit;
27     private JButton btnEntry;
28     private JSeparator spl;
29     private JButton btnCreate;
30
31     /**
32      * The user that logged in
33      */
34     private User user;
35
36     /**
```

```
37     * Constructs a new WarehouseToolbar object
38     * @param user - the user that logged in
39     */
40    public WarehouseToolbar(User user) {
41        this.user = user;
42        this.setBounds(0, 0, 392, 29);
43        this.setEnabled(false);
44        this.setVisible(true);
45    {
46        //Construct items only relevant to user permissions
47        if(user.isEntry() || user.isAdmin()){
48            btnCreate = new JButton();
49            this.add(btnCreate);
50            btnCreate.setIcon(new ImageIcon(getClass().getClassLoader()
51                .getResource(ApplicationConstants.CREATE_ITEM_ICON)));
52            btnCreate.setBounds(39, 85, 44, 38);
53            btnCreate.setToolTipText("Create a New Item");
54            btnCreate.setPreferredSize(new java.awt.Dimension(28, 28));
55            btnCreate.setSize(45, 28);
56            btnCreate.setOpaque(false);
57            btnCreate.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
58            btnCreate.addActionListener(new ActionListener(){
59                public void actionPerformed(ActionEvent e){
60                    ChangeScreen.setCreateNewItemScreen(getUser());
61                }
62            });
63        }
64    }
65    {
66        btnSearch = new JButton();
67        this.add(btnSearch);
68        btnSearch.setPreferredSize(new java.awt.Dimension(28, 28));
69        btnSearch.setIcon(new ImageIcon(getClass().getClassLoader()
70            .getResource(ApplicationConstants.ITEM_ENQUIRY_ICON)));
71        btnSearch.setBorder(BorderFactory.createCompoundBorder(null, null));
72        btnSearch.setFocusable(false);
```

```
73     btnSearch.setOpaque(false);
74     btnSearch.setSize(28, 28);
75     btnSearch.addActionListener(new ActionListener() {
76         public void actionPerformed(ActionEvent e){
77             ChangeScreen.setItemSearchScreen(getUser());
78         }
79     });
80 }
81
82 {
83     //Construct items only relevant to user permissions
84     if(user.isEntry() || user.isAdmin()){
85         btnEntry = new JButton();
86         this.add(btnEntry);
87         btnEntry.setIcon(new ImageIcon(getClass().getClassLoader()
88             .getResource(ApplicationConstants.ITEM_ENTRY_ICON)));
89         btnEntry.setBorder(BorderFactory.createCompoundBorder(null,
90             BorderFactory.createEmptyBorder(0, 0, 0, 0)));
91         btnEntry.setSize(28, 16);
92         btnEntry.setPreferredSize(new java.awt.Dimension(28, 10));
93         btnEntry.setOpaque(false);
94         btnEntry.addActionListener(new ActionListener(){
95             public void actionPerformed(ActionEvent e){
96                 ChangeScreen.setMultipleItemSearchScreen(getUser(), ApplicationConstants.ITEM_ENTRY);
97             }
98         });
99     }
100 }
101
102 {
103     //Construct items only relevant to user permissions
104     if(user.isExit() || user.isAdmin()){
105         btnExit = new JButton();
106         this.add(btnExit);
107         btnExit.setIcon(new ImageIcon(getClass().getClassLoader()
108             .getResource(ApplicationConstants.ITEM_EXIT_ICON)));

```

```
109     btnExit.setOpaque(false);
110     btnExit.setPreferredSize(new java.awt.Dimension(28, 10));
111     btnExit.setBorder(BorderFactory.createCompoundBorder(null, null));
112     btnExit.addActionListener(new ActionListener(){
113         public void actionPerformed(ActionEvent e){
114             ChangeScreen.setMultipleItemSearchScreen(getUser(), ApplicationConstants.ITEM_EXIT);
115         }
116     });
117 }
118 }
119 {
120     sp1 = new JSeparator();
121     this.add(sp1);
122     sp1.setSize(25, 25);
123     sp1.setPreferredSize(new java.awt.Dimension(25, 25));
124     sp1.setOrientation(SwingConstants.VERTICAL);
125 }
126 }
127 }
128 /**
129 * Gets the user that logged in
130 * @return the user that logged in
131 */
132 public User getUser(){
133     return user;
134 }
135 }
136 }
137 }
```

---

## C2 –Handling Errors

---

This section provides further explanation regarding the various methods of error handling used throughout the application. For each, the potential error and solution are explained, and a code excerpt is provided (only the directly relevant parts are quoted so as to avoid constant repetition). Error handling routines are examined class by class.

In addition to the potential errors and their solutions described below, generic errors due to wrong formats other than the validation rules outlined in section A2, are handled by setting limits and restrictions to GUI components that allow user input (mostly JTextFields). The class TextFieldLimit sets a maximum length to a textfield; the class TextFieldDigitLimit sets a maximum length of digits to a textfield. This prevents errors filtering to processing algorithms described below.

Two of several error handling routines of this type are provided below:

*Excerpts*

a. From the class CreateNewItem, line179 in page 123

For the item description field, the user is only given 200 characters. To handle potential errors due to violation of this validation rule, a TextFieldLimit document with an upper limit of 200 characters is set to the textfield.

```
txtDescription.setDocument(new TextFieldLimit(200));
```

b. From the class CreateNewItem, line 401 in page 130

For the item code field, the user is only allowed a 6 digit code. To handle potential errors due to violation of this validation rule, a TextFieldDigitLimit document with an upper limit of 6 digit (no other characters allowed) is set to the textfield.

```
txtCode.setDocument(new TextFieldDigitLimit(6));
```

The TextFieldLimit class is found in page 359.

*Excerpt*

From the class TextFieldLimit, lines 31-39 on page 357

```
/**  
 * Only inserts a character if the maximum number of characters has not been reached  
 */  
public void insertString (int offset, String str, AttributeSet attr) throws BadLocationException {  
    if (str == null) return;  
    if ((getLength() + str.length()) <= limit) {  
        super.insertString(offset, str, attr);  
    }  
}
```

The TextFieldDigitLimit class is found in page 357.

*Excerpt*

From the class TextFieldDigitLimit, lines 31-39 on pages 357-358

```
/**  
 * Only inserts a character if it is a digit  
 */  
public void insertString (int offset, String str, AttributeSet attr) throws BadLocationException {  
    if (str == null) return;  
  
    Character c = new Character(str.charAt(0));  
  
    if ((getLength() + str.length()) <= limit && Character.isDigit(c.charValue()) == true) {  
        super.insertString(offset, str, attr);  
    }  
}
```

## ACCOUNT SETTINGS

The screen that allows a user to change his/her password

### 1. Users file not found

A try/catch block surrounds the accessing of the Users file, so as to catch any IOException; the file may be non-existent or corrupt. If so, the user is informed by a dialog that the password change operation has not been completed.

*Excerpt*

From class AccountSettings, lines 144-162 in pages 112-113

```
try{
    [...]
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "The password has not been changed. \n" +
        "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);
    this.dispose();
}
```

### 2. The user's current password is not equal to the password input in the 'Current password' textfield

### 3. The confirmation of the new password is not equal to the new password

An if statement is written to prevent these two errors. If the information entered by the user is correct, the password change operation proceeds, else the user is informed by a dialog that the information is not correct.

*Excerpt*

From class AccountSettings, lines 145-156 in page 113

```
if(user.getPassword().equals(currentPassword) && (newPassword.equals(confirmPassword))){
    performChange(newPassword);
    JOptionPane.showMessageDialog(this, "Password successfully changed",
        "Information", JOptionPane.INFORMATION_MESSAGE);
    this.dispose();
```

```
    } else {
        JOptionPane.showMessageDialog(this, "Some fields are missing or the \ninformation " +
            "entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);
        txtCurrentPassword.setText("");
        txtNewPassword.setText("");
        txtConfirmPassword.setText("");
    }
}
```

#### 4. The tokenized string storing the User in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 7 tokens (for the 7 attributes of a user) is put into place. If the string has 7 tokens, the user is created, else a null user is returned.

*Excerpt*

From the class AccountSettings, lines 226-238 in page 115

```
if (tokenizer.countTokens() == 7){      //preliminary error checking
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    String fullName = tokenizer.nextToken();
    boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());

    return new User(username, password, fullName, admin, entry, exit, userEnabled);
}

return null;
```

## CREATE NEW ITEM

The screen that allows a user to create a new item.

#### 1. The information entered in the fields of the new item is incorrect or incomplete

When the user hits the create new item button, the information entered by him/her may be wrong. To prevent this, an if statement calls a method that returns true if the information entered is correct. If so, it proceeds to create the new item, and informs the user that the operation has been successful, else it displays an input error.

*Excerpts*

a. From class CreateNewItem, lines 253-261 in page 126:

```
public void actionPerformed (ActionEvent e){  
    if(isCorrectInput()){  
        [...]  
        displaySuccessMessage();  
    } else {  
        displayInputError();  
    }  
}
```

b. From class CreateNewItem, lines 498-511 in pages 132-133:

To check that the information entered is complete, an if statement makes sure that the required fields (name and code) are not empty or null and that the length of the code is 6. If so, it returns true, else false.

```
/**  
 * Validates the user input  
 * @return true if input is valid, else false  
 */  
public boolean isCorrectInput(){  
  
    if(!txtName.getText().equals("") || txtName.getText() != null  
        && txtCode.getText().length() == 6){  
        return true;  
    } else {  
        return false;  
    }  
}
```

}

c. From class CreateNewItem, lines 451-458 in page 131

If the information entered is complete and correct, the user is informed by a dialog that the item creation operation has been completed.

```
/**  
 * Informs the user that the item has been created  
 */  
public void displaySuccessMessage(){  
    JOptionPane.showMessageDialog(this, "The item has been created.",  
        "Success", JOptionPane.INFORMATION_MESSAGE);  
    ChangeScreen.setBlankScreen(user);  
}
```

d. From class CreateNewItem, lines 444-449 in page 131

If the information entered is not complete and correct, the user is informed by a dialog that the item creation operation cannot follow as some fields are missing or are incorrect.

```
/**  
 * Informs the user that the input is invalid or incorrect  
 */  
public void displayInputError(){  
    JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);  
}
```

**2. A null image or a corrupt image is chosen in the FileChooser**

An if statement checking that the image is not null and a try/catch block to prevent any IOExceptions (e.g. the image cannot be read) is implemented to avoid this error.

*Excerpt*

From class CreateNewItem, lines 472-486 in page 132

```
if(fileChooser.getSelectedFile() != null){
    File file = fileChooser.getSelectedFile();
    this.imageFile = file;
    try{
        Image image = ImageIO.read(file);
        lblItemImage.setIcon(new ImageIcon(image));
        btnClearImage.setEnabled(true);

    } catch (Exception e){

    }
}
```

**3. Items file not found or corrupt**

A try/catch block surrounds all the code that accesses the Items file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From class CreateNewItem, lines 549-610 in pages 134-135

```
try{
    RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
    [...]
} catch (Exception e){
    JOptionPane.showMessageDialog(
        MainScreen.contentPane, "One or files cannot be accessed. \n"
        + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

**4. Writing an item to a file that has not been initialised**

To prevent this error, an if statement checks that the size of the file is not 0. If it is, a method to initialize the file is called.

*Excerpt*

From class CreateNewItem, lines 552-554 in page 134

```
if(itemsFile.length() == 0){  
    initItemsFile(itemsFile);  
}
```

## 5. Writing an item to a file that is full

A while loop scans the file to find the position of the first empty item (whose ID is -999). If this position is greater than or equal to the file's length, a method is called to grow the file.

*Excerpt*

From class CreateNewItem, lines 560-573 in page 134

```
while(itemsFile.readInt() != -999 && !isFull){  
    emptyRecordPos = emptyRecordPos+1;  
    itemsFile.seek(emptyRecordPos * ApplicationConstants.ITEMS_FILE_RECORD_LENGTH);  
    if(emptyRecordPos >= (  
        itemsFile.length() / ApplicationConstants.ITEMS_FILE_RECORD_LENGTH) ) {  
        isFull = true;  
    }  
}  
  
if(isFull){  
    growItemsFile(itemsFile);  
}
```

## 6. Copying a null image to the item images directory

An if check is used to prevent this error. If the image is null, it is not copied.

*Excerpt*

From class CreateNewItem, lines 589-592 in page 135

```
if(imageFile!=null){  
    moveFile(this.imageFile);  
}
```

**7. Creating an index to a file that does not exist or is corrupt**

A try/catch block surrounds all the code that accesses the Index files, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From class CreateNewItem, lines 749-818 in pages 139-141

```
try{  
    File file = new File(ApplicationConstants.CODE_INDEX_FILE);  
    [...]  
} catch (Exception e){  
    JOptionPane.showMessageDialog(  
        MainScreen.contentPane, "One or files cannot be accessed. \n"  
        + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);  
}
```

## EDIT USER

The screen that allows an admin to change user information

**1. The information is incomplete**

If either the username, password or fullName is null and no permissions have been selected, the user cannot be updated. To prevent this error, an if statement checks that the information is full and complete. If not, the user is informed.

*Excerpt*

From the class EditUser, lines 250-265 in pages 149-150

```
if(txtUsername.getText() != null && txtPassword.getText() != null && txtFullName.getText() != null  
    && (chkAdmin.isSelected() || chkEntry.isSelected() || chkExit.isSelected())){  
    [...]  
} else {  
    JOptionPane.showMessageDialog(this, "Some fields are missing or the \n" +  
        "information entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);  
}
```

## 2. The users file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the Users file, so as to catch any IOException; the file may be non-existent or corrupt. If an exception is caught, the user is informed that the user update operation has not followed through in the form of a dialog. If not, the user is alerted that the operation has been successful. One of several code excerpts is provided below.

*Excerpt*

From the class EditUser, lines 252-261 in page 150

```
try{  
    [...]  
    JOptionPane.showMessageDialog(this, "The user has been updated",  
        "Information", JOptionPane.INFORMATION_MESSAGE);  
    [...]  
} catch(Exception e) {  
    JOptionPane.showMessageDialog(this, "The user has not been updated. \n" +  
        "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);  
    this.dispose();  
}
```

## 3. The tokenized string storing the User in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 7 tokens (for the 7 attributes of a user) is put into place. If the string has 7 tokens, the user is created, else a null user is returned.

*Excerpt*

From the class EditUser, lines 331-345 in page 152

```
if (tokenizer.countTokens() == 7){      //preliminary error checking
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    String fullName = tokenizer.nextToken();
    boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());

    return new User(username, password, fullName, admin, entry, exit, userEnabled);
}

return null;
```

## INDEX

The object that stores the index of an item

### 1. The index is compared to an object that is not an Index

An if statement checks that the object being compared is an instance of Index. If not, a ClassCastException is thrown. If it is an instance of Index, the comparison proceeds.

*Excerpt*

From the class Index, lines 110-112 in page 158

```
if (!(o instanceof Index)) {
    throw new ClassCastException();
}
```

## INDEXBTREE

The class that implements a Binary Tree of item indexes

### 1. The root (and consequently the entire tree) is deleted

To avoid this error, a dummy root is assigned as the root of the tree by using the first letter of the first node to be added to the tree.

*Excerpt*

From the class IndexBTree, lines 89-97 in page 161

```
if(isEmpty()) {
    //Add dummy root to avoid the root (and consequently the entire tree) being deleted
    String field = index.getIndex().getField().substring(0, 1);
    IndexTNode dummyRoot = new IndexTNode(new Index(-1, field));
    root = dummyRoot;
    insertIndex(index);
} else {
    insertIndex(root, index);
}
```

### 2. Adding an index that already has been added to the tree

An if statement checks that the node is not already in the tree. If it is, it is not added.

*Excerpt*

From the class IndexBTree, lines 107-110 in pages 161-162

```
if(index.getIndex().getField().equals(current.getIndex().getField()) &&
   index.getIndex().getPosition() == (current.getIndex().getPosition())){
    //Error checking: if the Index already exists, do nothing
    return;
```

### 3. Getting the node of a null pointer

In the getNode() method, an if statement checks that the pointer to the current node is not null. If it is, null is returned. If not, the recursive calls continue until the node is found.

*Excerpts*

From the class IndexBTree, lines 143-153 in pages 162-163

```
if (current == null) {
    return null;
} else {
    [...]
}
```

### 4. Searching for a node in a null pointer

In the search() and partialSearch() methods, an if statement checks that the pointer to the current node is not null. If it is, null is returned. If not, the recursive calls continue until the desired items are found. One of several error handling routines of this type is provided below.

*Excerpt*

From the class IndexBTree, lines 172-183 in pages 163-164

```
if (current == null) {
    return null;
} else {
    [...]
}
```

### 5. Getting results from a null sub-tree

An if statement checks that the sub-tree is not null. If it is, null is returned. If it is not, the appropriate search results are returned by calling getResults()

*Excerpt*

From the class IndexBTree, lines 197-201 in page 164

```
if(partialTree == null) {
    return null;
} else {
    return getResults(partialTree, query);
}
```

## 6. Trimming and returning an empty ArrayList of search results

An if statement checks that the ArrayList is not empty. If so, it is trimmed and returned, else null is returned.

*Excerpt*

From the class IndexBTree, lines 273-278 in page 166

```
if(!searchResults.isEmpty()) {
    searchResults.trimToSize();
    return searchResults;
} else {
    return null;
}
```

## 7. The dummy root is part of the search results sub-tree

Error checking to prevent the dummy root from being added to the ArrayList of search results is put in place by checking that its position is not -1. If it is the dummy root, the method is called on its children.

*Excerpt*

From the class IndexBTree, lines 288-294 in pages 166-167

```
if(current == null){  
    return;  
} else if(current.getIndex().getPosition() == -1){  
    //If dummy root, recursively call methods on children  
    addResults(current.getLeft(), query, searchResults);  
    addResults(current.getRight(), query, searchResults);  
} else if (!current.getIndex().getField().toLowerCase().startsWith(query.toLowerCase())) {
```

## ITEM

The object that stores data of an item in the warehouse

### 1. Writing to a non-existent or corrupt Items file

A try/catch block surrounds all the code that accesses the Items file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class Item, lines 265-260 in page 180

```
private void writeIDToFile(RandomAccessFile file){  
    try{  
        [...]  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(  
            MainScreen.contentPane, "One or files cannot be accessed. \n"  
            + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

### 2. The item's code is not 6 digits long

A while loop appends a 0 to the beginning of the code until its length is 0 to avoid this error.

*Excerpt*

From the class Item, lines 280-282 in page 180

```
while(code.length() != 6) {
    code = "0" + code;
}
```

**3. The item's name is not 200 characters long**

**4. The item's description is not 200 characters long (or is blank)**

A while loop appends a blank space to the end of these fields so that these validation rules are enforced. This makes sure that the record structure remains constant and the fields do not get mixed up for direct access. One of several code excerpts is provided below.

*Excerpt*

From the class Item, lines 298-300 in page 181

```
while(name.length() != 200) {
    name = name + " ";
}
```

## ITEM DESCRIPTION

**1. Loading a non-existent item image**

A try/catch block prevents a null pointer exception if the item image does not exist. If so, the 'no-image' picture is set to the label.

*Excerpt*

From the class ItemDescription, lines 456-465 in page 197

```
try {
    lblImage.setIcon(
```

```
        new ImageIcon(getClass().getClassLoader().getResource(
            "Files/Item_Images/" + item.getID() + ".jpg")));
    } catch (Exception e) {
        //Item has no Image
        lblImage.setIcon(
            new ImageIcon(getClass().getClassLoader().getResource(
                "Files/Item_Images/NoImage.jpg")));
    }
}
```

## 2. Writing to a non-existent or corrupt Items file

A try/catch block surrounds all the code that accesses the Items file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class ItemDescription, lines 821-839 in pages 207-208

```
try{
    RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
    [...]
} catch (Exception e){

}
```

## 3. Deleting a non-existent item image or transaction file

To prevent null pointer exceptions because of these errors, a try/catch block attempts to delete these files. If one or both does not exist, an exception is caught.

*Excerpt*

From the class ItemDescription, lines 1037-1045 in pages 213-214

```
try {
    File transactions = new File(
```

```
        ApplicationConstants.TRANSACTONS_FOLDER + item.getID() + ".txt");
transactions.delete();
File image = new File(ApplicationConstants.ITEM_IMAGES_FOLDER + item.getID() + ".jpg");
image.delete();
} catch (Exception e) {
    //One of the files may not exist
}
```

#### 4. Updated item information is incorrect or incomplete

A method returns a boolean that stores if the information is correct by having an if check to see if the required name and code fields are not empty.

*Excerpt*

From the class ItemDescription, lines 1048-1053 in page 214

```
if (!txtName.getText().equals(""))
    || txtName.getText() != null && txtCode.getText().length() == 6) {
    return true;
} else {
    return false;
}
```

#### 5. Accessing a non-existent or corrupt transactions file

A try/catch block surrounds all the code that accesses the Transactions file for the item, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class ItemDescription, lines 1217-1237 on pages 218-219

```
try{
    File file = new File(ApplicationConstants.TRANSACTONS_FOLDER + item.getID() + ".txt");
    [...]
```

```
} catch (Exception e){  
    JOptionPane.showMessageDialog(  
        MainScreen.contentPane, "One or files cannot be accessed. \n"  
        + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);  
}
```

## ITEMENTRY

The screen that allows a user to input the quantities of the unit of measurement to be entered into the warehouse of the one or more items that he/she searched for

- 1. The information necessary for an entry transaction (proof of reception and third party) is incomplete**
- 2. The quantity to enter is less than or equal to 0**

An if statement that calls a method to check that the input is correct prevents these errors.

### *Excerpts*

- a. From the class ItemEntry, lines 250-257 in pages 234-235

If the information is correct, the processing of the entry transaction follows, else the user is informed that some fields are missing or incorrect

```
if(isCorrectInput()){  
    processAllTransactions();  
    JOptionPane.showMessageDialog(this, "Transactions successfully processed",  
        "Success", JOptionPane.INFORMATION_MESSAGE);  
    ChangeScreen.setBlankScreen(user);  
} else {  
    JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);  
}
```

b. From the class ItemEntry, lines 285-302 in pages 235-236

If checks for necessary conditions

```
public boolean isCorrectInput(){
    TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
    boolean isComplete = true;

    for(int i=0;i<model.getRowCount();i++){
        TransactionRecord record = model.getTransactions()[i];
        if(record.getQuantity() <= 0){
            isComplete = false;
            break;
        }
    }

    if(txtProof.getText().equals("") || txtDeliveredBy.equals("") || !isComplete){
        return false;
    } else {
        return true;
    }
}
```

## ITEM EXIT

The screen that allows a user to input the quantities of the unit of measurement to be removed from the warehouse of the one or more items that he/she searched for

1. The information necessary for an entry transaction (exit voucher and third party) is incomplete
2. The quantity to remove is greater than the quantity in stock or less than or equal to 0

An if statement that calls methods to check that the input is correct prevents these errors.

*Excerpts*

a. From the class ItemExit, lines 246-261 in pages 243-244

If statements that check that both conditions are met and inform the user what the error is if they are not

```
public void performTransactionProcessing(){
    if(isCompleteInput() && isCorrectQuantity()){
        processAllTransactions();
        JOptionPane.showMessageDialog(this, "Transactions successfully processed",
            "Success", JOptionPane.INFORMATION_MESSAGE);
        ChangeScreen.setOptimalRouteScreen(user, itemsToProcess);
    } else if(!isCorrectQuantity() && !isCompleteInput()){
        JOptionPane.showMessageDialog(this, "Some fields are missing and the \n"
            + "quantity entered is incorrect", "Error", JOptionPane.ERROR_MESSAGE);
    } else if(!isCompleteInput()){
        JOptionPane.showMessageDialog(this, "Some fields are missing", "Error", JOptionPane.ERROR_MESSAGE);
    } else if(!isCorrectQuantity()){
        JOptionPane.showMessageDialog(this, "One or more quantities entered are incorrect",
            "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

b. From the class ItemExit, lines 306-312 in page 245

If checks to see that the necessary fields are complete.

```
public boolean isCompleteInput(){
    if(txtVoucher.getText().equals("") || txtRequestedBy.getText().equals("")){
        return false;
    } else {
        return true;
    }
}
```

c. From the class ItemExit, lines 288-300 in pages 244-245

Loop and if checks to see if the quantities entered are correct

```
public boolean isCorrectQuantity(){
    TransactionProcessingTableModel model = (TransactionProcessingTableModel)tblQuantities.getModel();
    for(int i=0;i<model.getRowCount();i++) {
        TransactionRecord record = model.getTransactions()[i];
        Item item = (Item)model.getSearchResults().get(i);
        if(record.getQuantity() > Transactions.getQuantityInStock(item) || record.getQuantity() <= 0) {
            return false;
        }
    }

    return true;
}
```

## ITEMS

Class that contains methods relevant to item processing

### 1. Writing to a non-existent or corrupt Items file

A try/catch block surrounds all the code that accesses the Items file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class Items, lines 25-59 in pages 246-247

```
try{
    RandomAccessFile itemsFile = new RandomAccessFile(ApplicationConstants.ITEMS_FILE, "rw");
    [...]
} catch (Exception e){
    return null;
}
```

}

## 2. Searching empty items

To avoid this error and source of inefficiency, an if statement checks that the ID of the item being searched by either group or location is not -999. One of several code excerpts is provided below.

*Excerpt*

From the class Items, lines 110-118 in page 249

```
if(position!=-999){           //if record is not empty
    [...]
}
```

## ITEMSEARCH

Screen that allows a user to search for an item by several search criteria

### 1. Code query is not 6 digits long or is empty

To avoid this error that would lead to wrong search results, if checks to verify the length of the code query are put in place. The TextFieldDigitLimit document is set to the textfield so further validation to see if the code query is only digits or longer than 6 digits in length is not necessary. If the conditions are met, the search by code proceeds.

*Excerpt*

From the class ItemSearch, lines 494-500 in page 264

```
if(!txtSearchByCode.getText().equals("")){
    if(txtSearchByCode.getText().length() != 6{
        JOptionPane.showMessageDialog(this, "You must enter a 6 digit code query",
            "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        performSearchByCode(txtSearchByCode.getText());
    }
}
```

}

## 2. Name query is empty

To avoid this error, if checks verify that the name query is not null or 0 in length. If so, the user is informed, else the search by name proceeds.

*Excerpt*

From the class ItemSearch, lines 506-515 in page 265

```
if(!txtSearchByName.getText().equals("")){  
    if(rbtnExactMatch.isSelected()){  
        performExactSearchByName(txtSearchByName.getText());  
    } else {  
        performPartialSearchByName(txtSearchByName.getText());  
    }  
} else {  
    JOptionPane.showMessageDialog(this, "You must enter a search query",  
        "Error", JOptionPane.ERROR_MESSAGE);  
}
```

## LOCATION

### 1. Aisle or column number is less than two digits in length

To avoid this error that would lead to an incorrectly formatted location of an item, an if check is put in place that checks the length of either of these fields. If it is less than 2, a 0 is appended to the front. One of several code excerpts is provided below.

*Excerpt*

From the class Location, lines 137-139 in page 273

```
if(aisle.length()!= 2){  
    aisle = "0" + aisle;
```

}

## 2. Compare a Location object to another object that is not an instance of Location

An if statement checks that the object that is being compared is an instance of Location. If so, the comparison follows through, else a ClassCastException is thrown.

*Excerpt*

From the class Location, lines 170-189 in page 274

```
if (o instanceof Location) {  
    Location location = (Location) o;  
    [...]  
} else {  
    throw new ClassCastException();  
}
```

## LOGIN SCREEN

The screen that allows a user to log in

### 1. The users file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the users file, so as to catch any IOException; the file may be non-existent or corrupt. If an exception is caught, a null user is returned. An if check alerts the user if this has happened, or else login proceeds.

*Excerpt*

a. From the class LoginScreen, lines 179-206 in pages 280-281

Try/catch block surrounding the accessing of the file

```
try{  
    File file = new File(ApplicationConstants.USERS_FILE);
```

```
[...]
} catch(Exception e){
    return null;
}
```

b. From the class LoginScreen, lines 156-166 in page 280

If check after login algorithm

```
if(user == null){
    JOptionPane.showMessageDialog(this, "Invalid username or password.",
        "Login Error", JOptionPane.ERROR_MESSAGE);
    [...]
} else {
    MainScreen main = new MainScreen(user);
    main.setVisible(true);
    this.dispose();
}
```

## 2. The tokenized string storing the User in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 7 tokens (for the 7 attributes of a user) is put into place. If the string has 7 tokens, the user is created, else a null user is returned.

*Excerpt*

From the class LoginScreen, lines 218-230 in page 282

```
if (tokenizer.countTokens() == 7){      //preliminary error checking
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    String fullName = tokenizer.nextToken();
    boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
```

```
        return new User(username, password, fullName, admin, entry, exit, userEnabled);  
    }  
  
    return null;  
}
```

## MAIN SCREEN

The landing screen of the application that loads up all utilities for operations

### 1. Accessing corrupt or non-existent Items, Groups, UMs or Index files

A try/catch block surrounds all the code that accesses these files, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class MainScreen, lines 148-183 in page 287

```
try{  
    File file = new File(ApplicationConstants.GROUPS_FILE);  
    [...]  
} catch(Exception e){  
    return null;  
}
```

### 2. Preventing NumberFormatExceptions when parsing the Group or UM code stored in the file

To prevent this error, a while loops removes leading 0s from either of these codes, and then the number is evaluated. One of several excerpts is provided below.

*Excerpt*

From the class MainScreen, lines 195-199 in page 288

```
while(code.startsWith("0")){
    code = code.substring(1, code.length());
}

short codeValue = new Integer(code).shortValue();
```

### 3. Inserting a null node into the binary tree

In the algorithm to add nodes to the binary trees so that the tree is balanced, an if check makes sure that the index of the node in the node array is not invalid. Only then does node insertion proceed.

*Excerpt*

From the class MainScreen, lines 296-305 in page 291

```
if(mid > indexes.length-1){
    return;
} else if (size == 0){
    tree.insertIndex(new IndexTNode(indexes[mid]));
} else {
    tree.insertIndex(new IndexTNode(indexes[mid]));
    createIndexBinaryTree(indexes, tree, start, mid);
    createIndexBinaryTree(indexes, tree, mid+1, finish);
}
```

## MANAGE USERS

The screen that allows an admin user to view users in a table and select to edit or delete one or more

### 1. Opening the edit user screen for a null pointer to a user in the table

An if check makes sure that the index of the selected row is not -1 (i.e. no row is selected) to prevent this error. Only then does the operation proceed.

*Excerpt*

From the class ManageUsers, lines 136-141 in page 297

```
if(tblUsers.getSelectedRow() != -1) {
    [...]
}
```

## 2. The users file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the users file, so as to catch any IOException; the file may be non-existent or corrupt. If an exception is caught, the user is informed. If not, the operation proceeds. One of several code excerpts is provided below.

*Excerpt*

From the class ManageUsers, lines 168-206 in pages 298-263

```
try{
    File file = new File(ApplicationConstants.USERS_FILE);
    [...]
    JOptionPane.showMessageDialog(this, "The user has been successfully deleted",
        "Information", JOptionPane.INFORMATION_MESSAGE);
} catch (Exception e){
    JOptionPane.showMessageDialog(this, "The user has not been deleted. \n"
        + "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);
}
```

## 3. The tokenized string storing the User in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 7 tokens (for the 7 attributes of a user) is put into place. If the string has 7 tokens, the user is created, else a null user is returned.

*Excerpt*

From the class ManageUsers, lines 263-275 in page 301

```
if (tokenizer.countTokens() == 7){      //preliminary error checking
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    String fullName = tokenizer.nextToken();
    boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());

    return new User(username, password, fullName, admin, entry, exit, userEnabled);
}

return null;
```

## MULTIPLE ITEM SEARCH

The screen that allows a user to search for multiple items to enter or withdraw from the warehouse

### 1. Proceed to transact an empty list of items

To avoid this error that would create a null pointer exception, an if check is added to the method called when the 'next step' button is pressed to verify that the list is not empty. If so, the operation proceeds, else the user is informed.

*Excerpt*

From the class MultipleItemSearch, lines 750-768 in pages 322-323

```
if(lstItems.getModel().getSize() == 0){
    JOptionPane.showMessageDialog(this, "You must add items to the list",
        "Error", JOptionPane.ERROR_MESSAGE);
} else {
    [...]
}
```

## 2. Adding a null item from an empty table of search results to the list of items to transact

To avoid this error, the button to add to the list is only enabled when a row of the table is selected. A method is called to verify this with an if check by getting the index of the selected row.

*Excerpts*

- From the class MultipleItemSearch, lines 167-169 in page 306

```
public void valueChanged(ListSelectionEvent e) {
    enableAdd(e.getFirstIndex());
}
```

- From the class MultipleItemSearch, lines 565-571 in page 317

```
public void enableAdd(int i) {
    if(i<0) {
        btnAdd.setEnabled(false);
    } else {
        btnAdd.setEnabled(true);
    }
}
```

## 3. Removing a null item from the list of items to transact

To avoid this error, the button to remove from the list is only enabled when an item in the list is selected. A method is called to verify this with an if check by getting the index of the selected item in the list.

*Excerpts*

- From the class MultipleItemSearch, lines 232-234 in page 308

```
public void valueChanged(ListSelectionEvent e) {
    enableRemove(e.getFirstIndex());
}
```

b. From the class MultipleItemSearch, lines 709-715 in page 321

```
public void enableRemove(int i){
    if(i<0){
        btnRemove.setEnabled(false);
    } else {
        btnRemove.setEnabled(true);
    }
}
```

#### 4. When processing item withdrawals, attempting to remove an item that is not in stock (i.e. quantity is 0)

When the user attempts to add an item to the list of items when processing a withdrawal request, an if check verifies that the quantity of the item to remove from the warehouse is greater than 0.

*Excerpt*

From the class MultipleItemSearch, lines 692-694 in page 321

```
} else if (quantity <= 0){
    JOptionPane.showMessageDialog(this, "The item is not in stock", "Error", JOptionPane.ERROR_MESSAGE);
}
```

## MULTIPLE SEARCH RESULTS TABLE MODEL

The table model for the table displaying multiple search results

### 1. Getting the size of a null ArrayList

When the query yields no results, a MultipleSearchResultsTableModel object is constructed with null as its parameter. To avoid a null pointer exception when attempting to get the size of this null ArrayList, an if check verifies if the ArrayList is null. If it is, the row count is 0, else it is the size of the ArrayList.

*Excerpt*

From the class MultipleSearchResultsTableModel, lines 40-47 in page 326

```
public int getRowCount() {
    if(searchResults!= null){
        return searchResults.size();
    } else {
        return 0;
    }
}
```

## NEWUSER

The screen that allows an admin user to create a new user

### 1. The users file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the users file, so as to catch any IOException; the file may be non-existent or corrupt. If an exception is caught, the user is informed. If not, the operation proceeds. One of several code excerpts is provided below.

*Excerpt*

From the class NewUser, lines 214-224 in pages 335-336

```
try{
    writeNewUser(user);
    JOptionPane.showMessageDialog(this, "The user has been created",
        "Information", JOptionPane.INFORMATION_MESSAGE);
```

```
        this.dispose();
    } catch(Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "The user has not been created. \n"
            + "The files may be corrupted", "Error", JOptionPane.ERROR_MESSAGE);
        this.dispose();
    }
}
```

## 2. Attempting to create a user with incomplete required information

To avoid this error, an if check is implemented to make sure that all the required fields have been completed. If not, the user is alerted.

*Excerpt*

From the class NewUser, lines 210-228 in pages 335-336

```
if(txtUsername.getText() != null && txtPassword.getText() !=null && txtFullName.getText() != null
    && (chkAdmin.isSelected() || chkEntry.isSelected() || chkExit.isSelected())){
    User user = new User(txtUsername.getText(),txtPassword.getText(), txtFullName.getText(),
        chkAdmin.isSelected(),chkEntry.isSelected(),chkExit.isSelected(), true);
    [...]
} else {
    JOptionPane.showMessageDialog(this, "Some fields are missing or the \n"
        + "information entered is erroneous.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

## 3. The tokenized string storing the User in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 7 tokens (for the 7 attributes of a user) is put into place. If the string has 7 tokens, the user is created, else a null user is returned.

*Excerpt*

From the class NewUser, lines 306-318 in page 338

```
if (tokenizer.countTokens() == 7){      //preliminary error checking
    String username = tokenizer.nextToken();
    String password = tokenizer.nextToken();
    String fullName = tokenizer.nextToken();
    boolean admin = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean entry = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean exit = intToBoolean(new Integer(tokenizer.nextToken()).intValue());
    boolean userEnabled = intToBoolean(new Integer(tokenizer.nextToken()).intValue());

    return new User(username, password, fullName, admin, entry, exit, userEnabled);
}

return null;
```

## SEARCHRESULTSTABLEMODEL

### 1. Getting the size of a null ArrayList

When the query yields no results, a SearchResultsTableModel object is constructed with null as its parameter. To avoid a null pointer exception when attempting to get the size of this null ArrayList, an if check verifies if the ArrayList is null. If it is, the row count is 0, else it is the size of the ArrayList.

*Excerpt*

From the class SearchResultsTableModel, lines 43-50 in page 353

```
public int getRowCount(){
    if (searchResults != null){
        return searchResults.size();
    } else {
        return 0;
    }
}
```

## TRANSACTIONLIST

- 1. Searching for a node in a null pointer**
- 2. Returning an empty ArrayList of search results**

To avoid this error, in the searchTransactions() method, an if statement checks that the pointer to the current node is not null. If it is, another if check verifies that the ArrayList of search results is not empty. Is so, the ArrayList is returned, else null is returned. If the pointer to the current node is not null, the recursive calls continue until the desired items are found.

*Excerpt*

From the class TransactionList, lines 87-103 in page 369

```
if(current == null){  
    if(searchResults.isEmpty()) {  
        return null;  
    } else {  
        return searchResults;  
    }  
} [...]
```

- 3. Getting the node of a null pointer**

In the getNode() method, an if statement checks that the pointer to the current node is not null. If it is, null is returned. If not, the recursive calls continue until the node is found.

*Excerpts*

From the class TransactionList, lines 122-124 in page 363

```
if (current == null){  
    return null;  
} else {  
    [...]
```

}

## TRANSACTION PROCESSING TABLE MODEL

The table model for the table of items selected by the user to transact

### 1. Entering an invalid quantity (i.e. non-integer) in the quantity row

To avoid this error, the class of that column in the table model is defined to be Integer. This does not allow the user to enter a non-integer value.

*Excerpt*

From the class TransactionProcessingTableModel, lines 128-146 in pages 371-372

```
public Class getColumnClass(int col){  
  
    switch(col){  
  
        [...]  
        case 4:  
            return Integer.class;  
        default:  
            return null;  
    }  
  
}
```

## TRANSACTION RECORD

The object that stores data of an item transaction (entry to or withdrawal from the warehouse)

### 1. The transactions file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the transactions file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class TransactionRecord, lines 242-291 in pages 380-381

```
try{
    File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + itemID + ".txt");
    [...]
} catch (Exception e){
    JOptionPane.showMessageDialog(
        MainScreen.contentPane, "One or files cannot be accessed. \n"
        + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

## TRANSACTIONS

The class that contains methods relevant to transactions processing

### 1. The transactions file does not exist or is corrupt

A try/catch block surrounds all the code that accesses the transactions file, so as to catch any IOException; the file may be non-existent or corrupt. One of several code excerpts is provided below.

*Excerpt*

From the class Transactions, lines 23-49 in pages 383-384

```
try{
    File file = new File(ApplicationConstants.TRANSACTIONS_FOLDER + itemID + ".txt");
    [...]
} catch (Exception e){
    JOptionPane.showMessageDialog(
        MainScreen.contentPane, "One or files cannot be accessed. \n"
        + "Contact the system administrator.", "Error", JOptionPane.ERROR_MESSAGE);
}
```

## 2. The tokenized string storing the TransactionRecord in the file is corrupt or incorrect

To prevent wrong information and null pointer exceptions, the string is tokenized and verification that it has 10 tokens (for the 10 attributes required to construct a TransactionRecord) is put into place. If the string has 10 tokens, the TransactionRecord is created, else null is returned.

*Excerpt*

From the class Transactions, lines 60-77 in pages 384-385

```
if (tokenizer.countTokens() == 10){      //preliminary error checking
    int ID = new Integer(tokenizer.nextToken()).intValue();
    int type = new Integer(tokenizer.nextToken()).intValue();
    int day = new Integer(tokenizer.nextToken()).intValue();
    int month = new Integer(tokenizer.nextToken()).intValue();
    int year = new Integer(tokenizer.nextToken()).intValue();
    int quantity = new Integer(tokenizer.nextToken()).intValue();
    int balance = new Integer(tokenizer.nextToken()).intValue();
    String warehouseWorker = tokenizer.nextToken();
    String document = tokenizer.nextToken();
    String thirdParty = tokenizer.nextToken();
    GregorianCalendar date = new GregorianCalendar(year, month, day);

    return new TransactionRecord(ID, type, date, quantity, balance, warehouseWorker,
        document, thirdParty, item);
}
return null;
```

# TRANSACTIONS TABLE MODEL

## 1. Getting the size of a null ArrayList

If an item has no transactions, a TransactionsTableModel object is constructed with null as its parameter. To avoid a null pointer exception when attempting to get the size of this null ArrayList, an if check verifies if the ArrayList is null. If it is, the row count is 0, else it is the size of the ArrayList.

*Excerpt*

From the class TransactionsTableModel, lines 43-47 in page 387

```
if(transactionsToDisplay!= null) {
    return transactionsToDisplay.size();
} else {
    return 0;
}
```

## C3 – Success of the Program

This section assesses the level of success of the application against the objectives set in criterion A2.

Objective	Test Reference (Page number)	Level of Success
1	<i>See below</i>	Full
1.1	<i>See below</i>	Full
1.1.1	492-510	Full
1.1.2	492-510	Full
1.1.3	492-510	Full
1.1.4	492-510	Full
1.1.5	492-510	Full
1.2	<i>See below</i>	Full
1.2.1	537-551	Full
1.2.2	537-551	Full
1.2.3	571-576	Full
1.3	<i>See below</i>	Full
1.3.1	514-535	Full
1.3.2	536	Full
2	<i>See below</i>	Full
2.1	615-625	Full
2.2	626-638	Full
2.3	558-562	Full
3	<i>See below</i>	Partial
3.1	583-605	Full
3.2	615-625	Full
3.3	639-642	Partial (not the BEST route)
3.4	626-638	Full
4	<i>See below</i>	Full
4.1	460-642	Full

4.2	460-642	Full
4.3	465-468	Full
4.4	465-468	Full
4.5	460-642	Full
5	<i>See below</i>	Full
5.1	<i>See below</i>	Full
5.1.1	460-464	Full
5.1.2	465-468	Full
5.1.3	470-471	Full
5.2	<i>See below</i>	Full
5.2.1	460-464	Full
5.2.2	465-468	Full
5.2.3	475-479	Full
5.2.4	485-489	Full
6	<i>See below</i>	Full
6.1	460-642	Full
6.2	460-642	Full
6.3	639-642	Full
6.4	460-642	Full

## Section D

---

# Documentation

## D1 – Test Output

This section should serve as proof for the various features of the application described in the previous sections. All screens have been tested for different input and output data sets.

### LOGIN

The login process to the application will be tested. Below is a screenshot of the users file that contains some users to demonstrate login functionality.



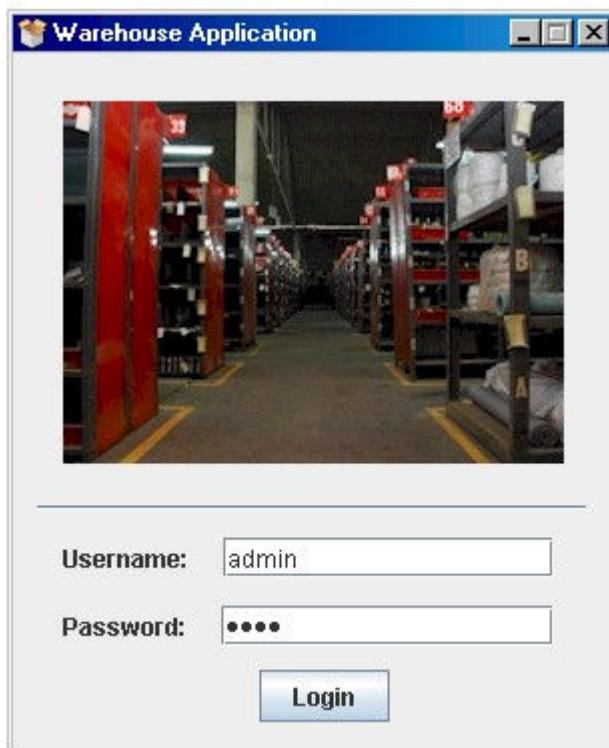
```
File Edit Format View Help
admin|root|System Administrator|1|1|1|1
entryGuy|password|Entry Guy|0|1|0|1
exitGuy|1234|Exit Guy|0|0|1|1
disabled|1234|New Name|0|0|1|1
nonAdmin|xyz|Non Admin|0|1|1|1
```

### CORRECT INPUT

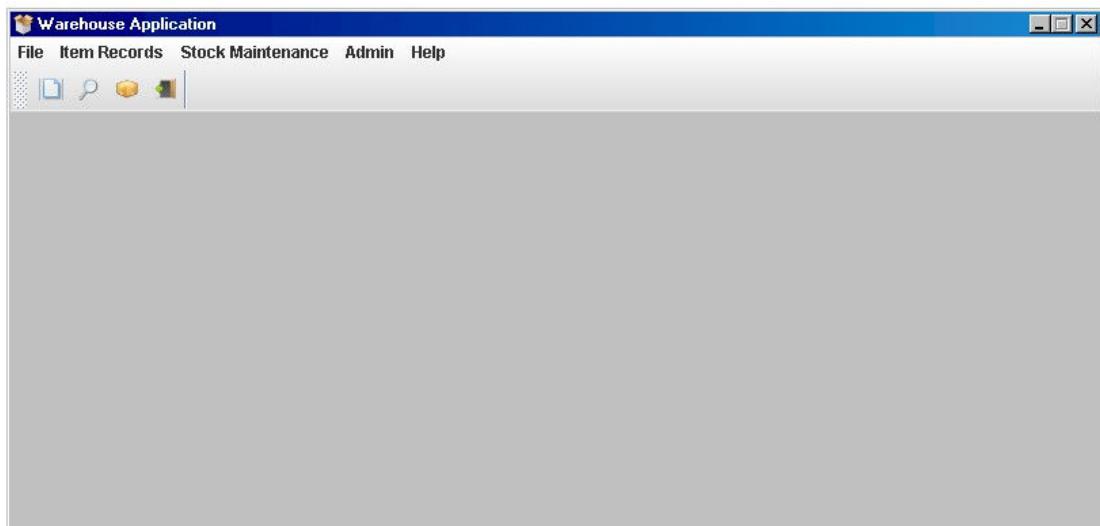
1. Existing username with correct password

*Test:*

Username: admin  
Password: root



*Result:* Login successful, application's landing screen loads

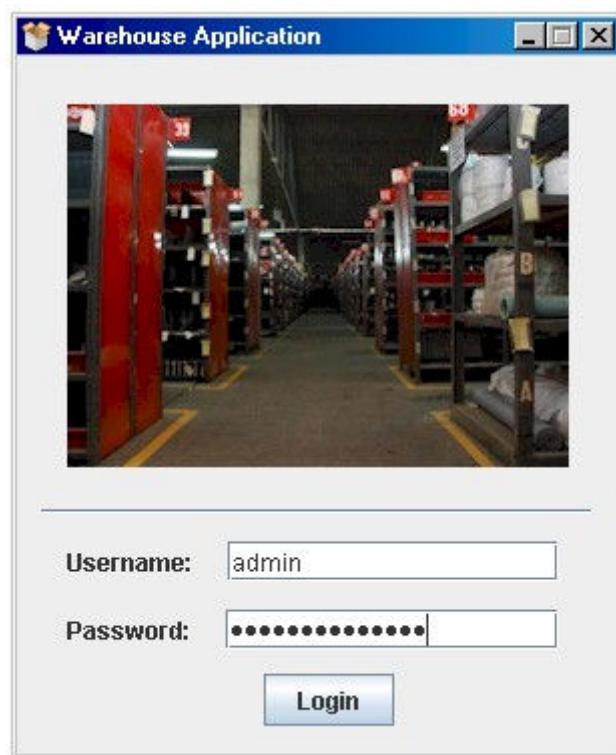


### **INCORRECT INPUT**

1. An existing username with an incorrect password  
This situation also applies for a username that does not exist

*Test:*

Username: admin  
Password: wrongPassword!



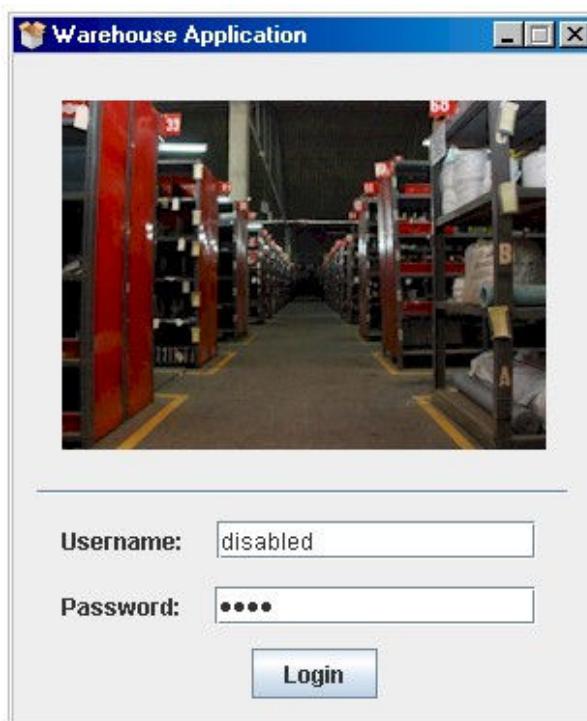
*Result:* Login unsuccessful, user alerted that username or password is invalid



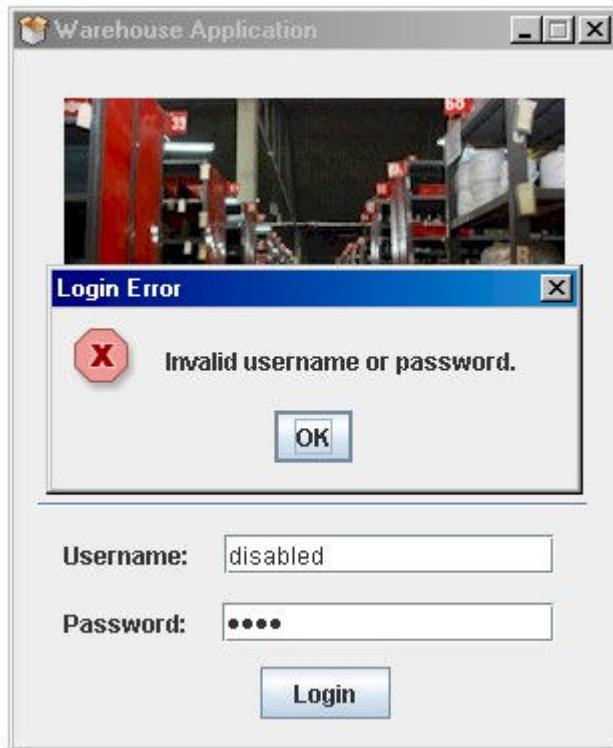
2. A disabled user attempts to log into the application

*Test:*

Username: disabled  
Password: abcd



*Result:* Login unsuccessful, user alerted that username or password is invalid



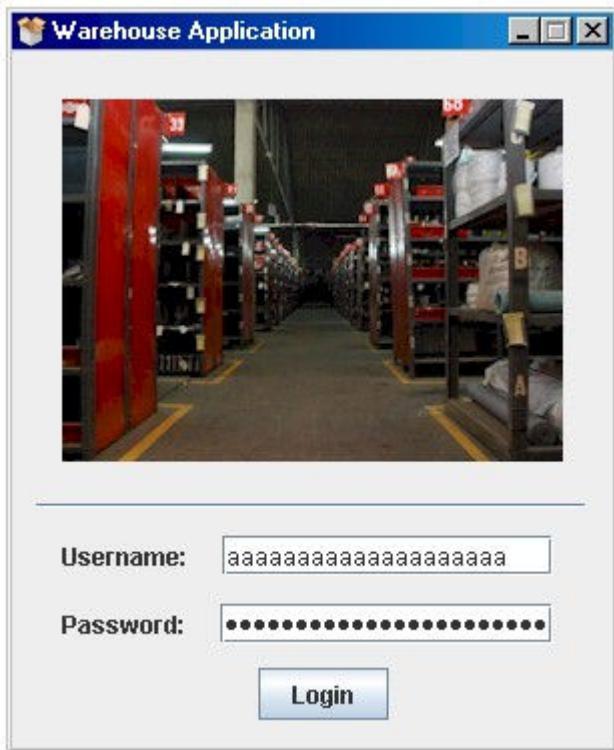
3. Username longer than 20 characters and password longer than 30 characters

*Test:*

Username:aaaaaaaaaaaaaaaaaaaaaaaaaaaa [25 characters]  
Password: bbbbbbbbbb bbbb bbbb bbbb bbbb bbbb [35 characters]

*Result:*

Textfields do not allow for more than 20 and 30 characters respectively, and truncate user input.



*Result:* Login unsuccessful, user alerted that username or password is invalid



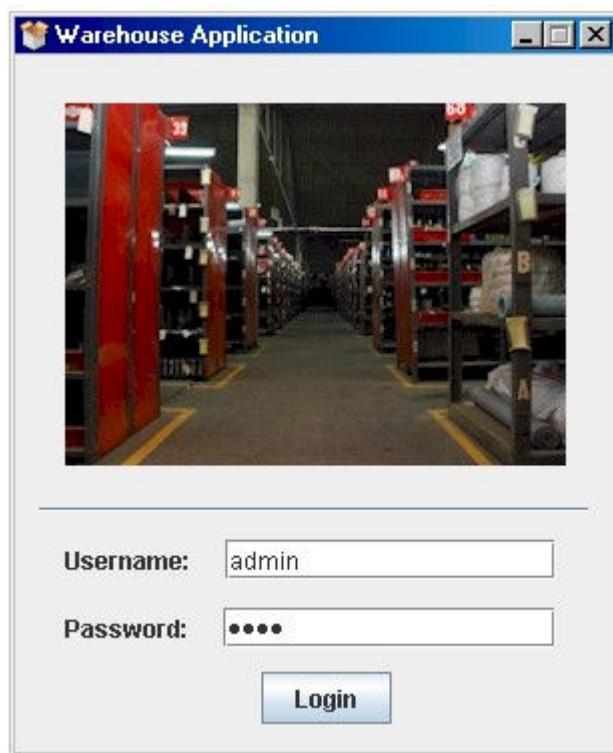
## LOADING APPROPRIATE USER PERMISSIONS

Users with different permissions only have access to functions directly related to their role. These tests document the different possibilities that may be displayed to the user. The same Users file shown in tests for Login is used.

1. User with admin permissions logs in

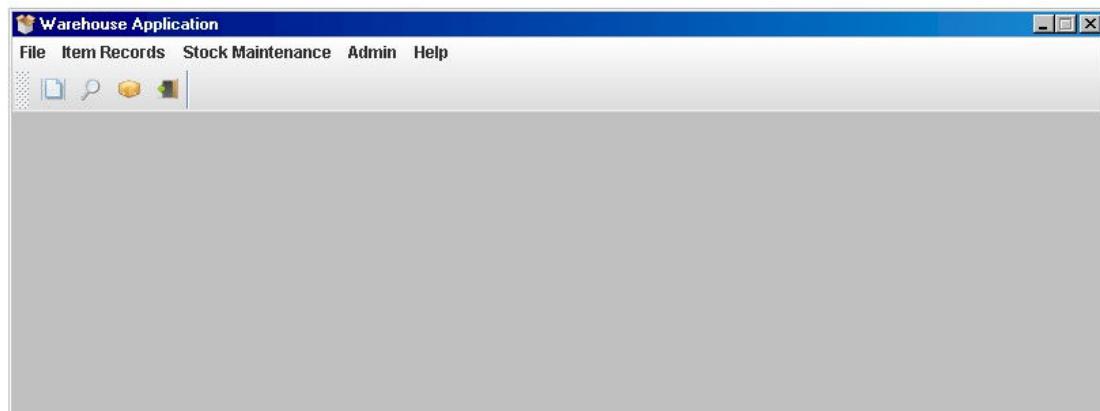
*Test:*

Username: admin  
Password: root



*Result*

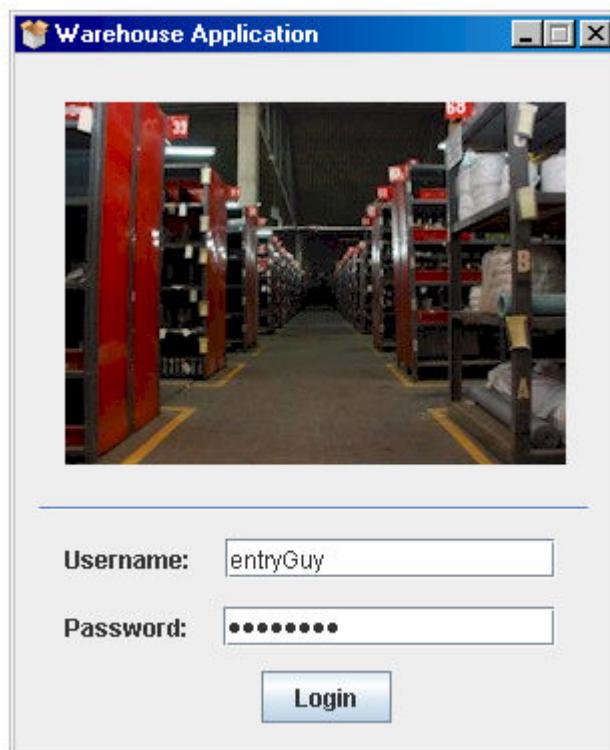
All functions in the toolbar and menu bar load and are accessible to this admin user.



2. User with only item entry permissions logs in

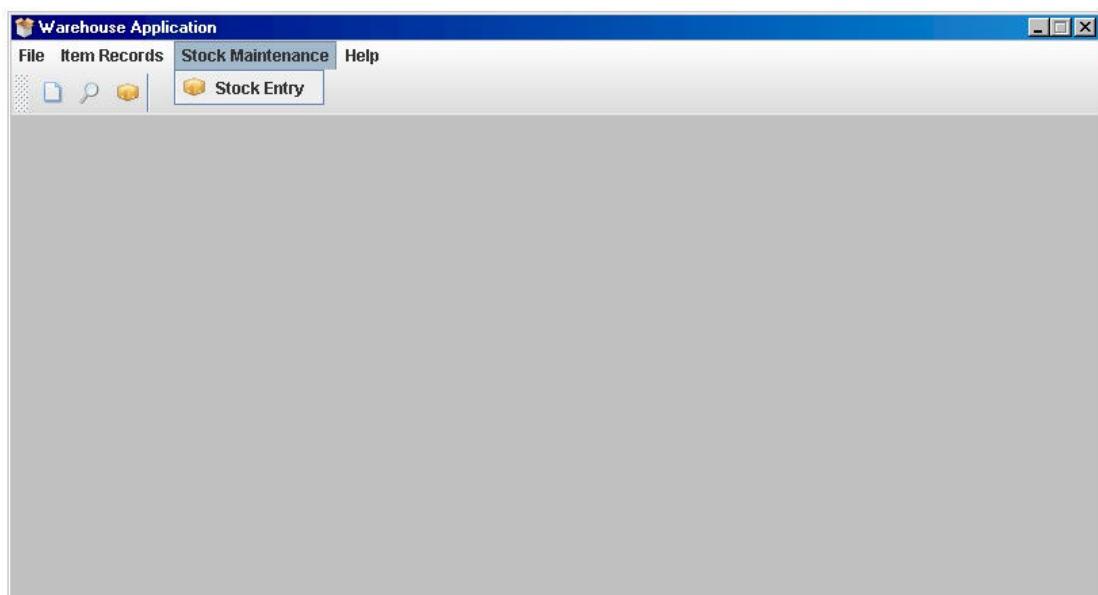
*Test:*

Username: entryGuy  
Password: password



*Result*

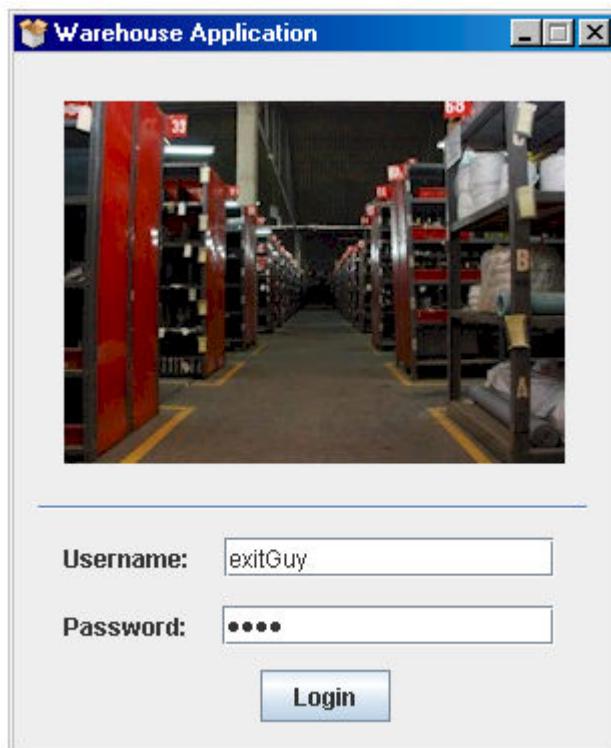
Only item entry-related functions are accessible to the user in the toolbar and menu bar (not item exit or admin functions)



3. User with only item exit permissions logs in

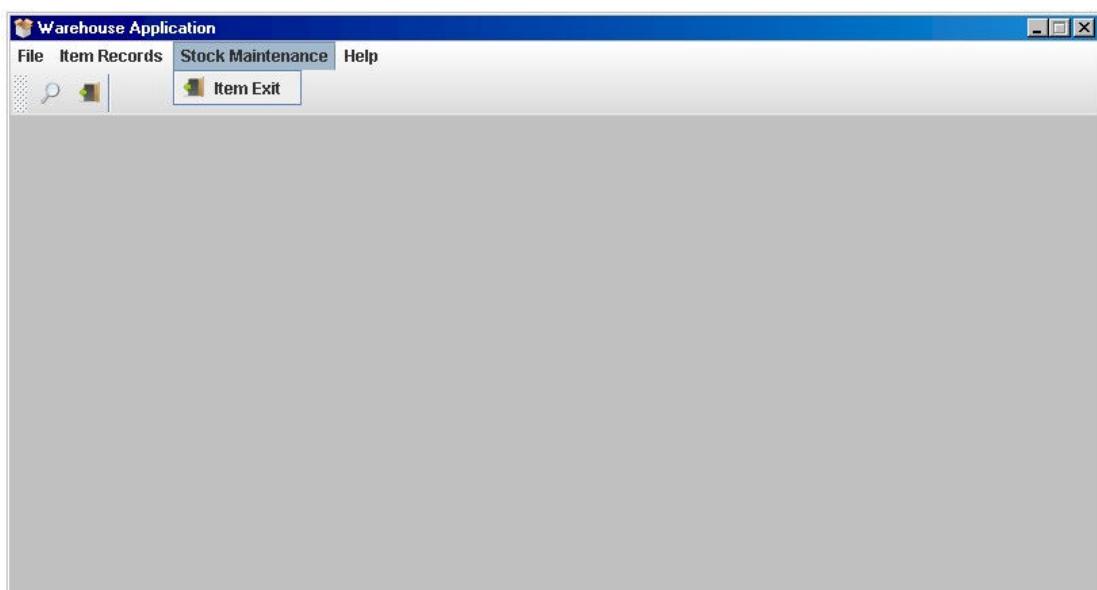
*Test:*

Username: exitGuy  
Password: 1234



*Result:*

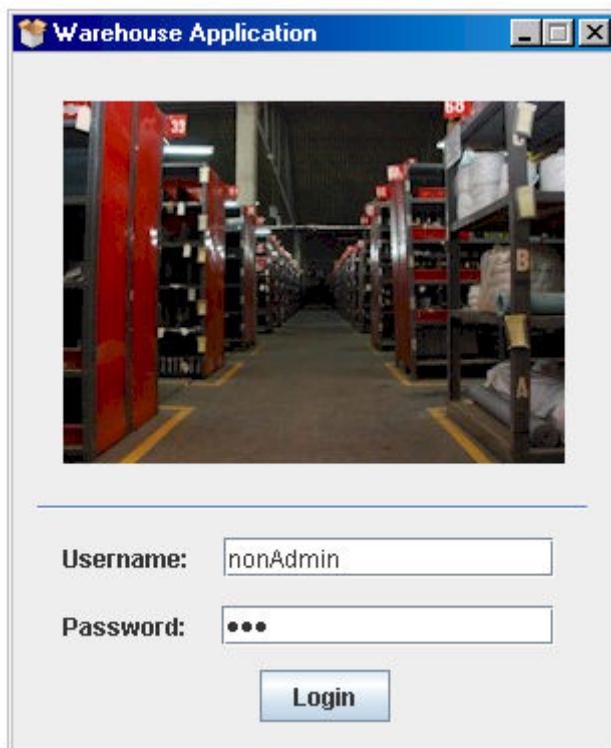
Only item exit-related functions are accessible to the user in the toolbar and menu bar (not item entry or admin functions)



4. User with only item entry and item exit permissions (not admin) logs in

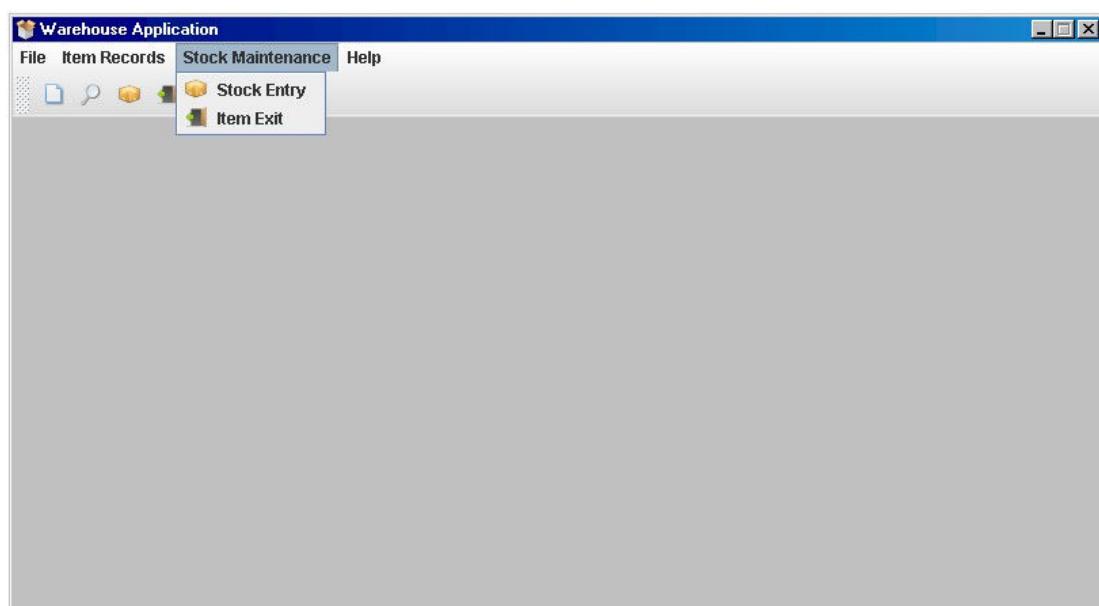
*Test:*

Username: nonAdmin  
Password: xyz



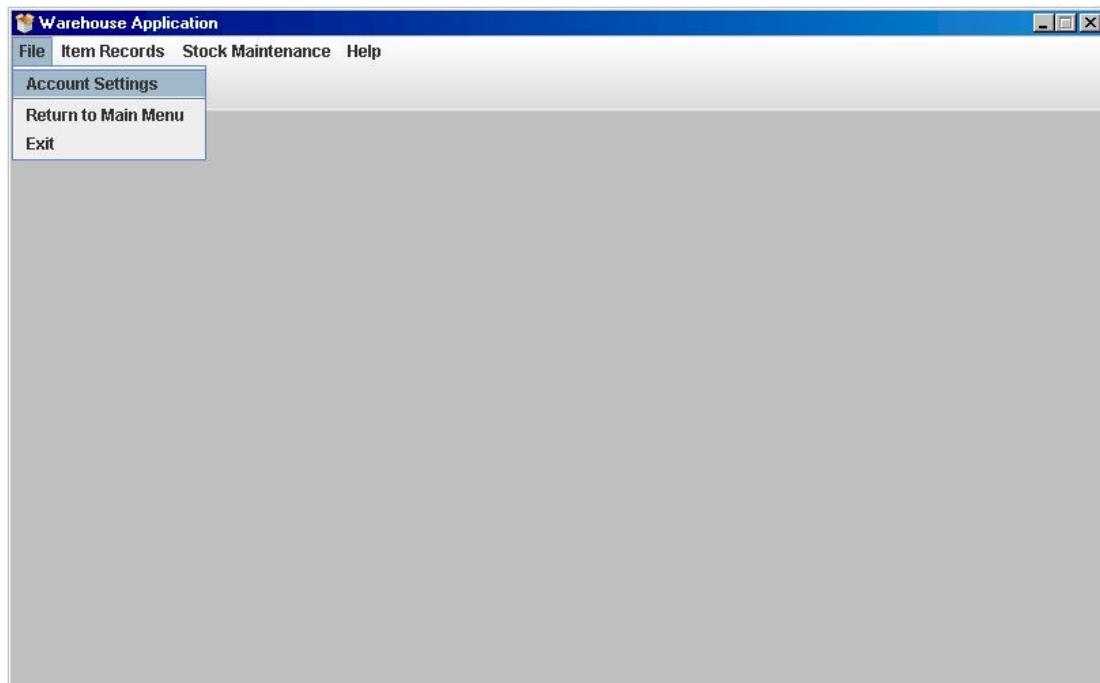
*Result:*

Only item entry and item exit-related functions are accessible to the user in the toolbar and menu bar (not admin related functions)



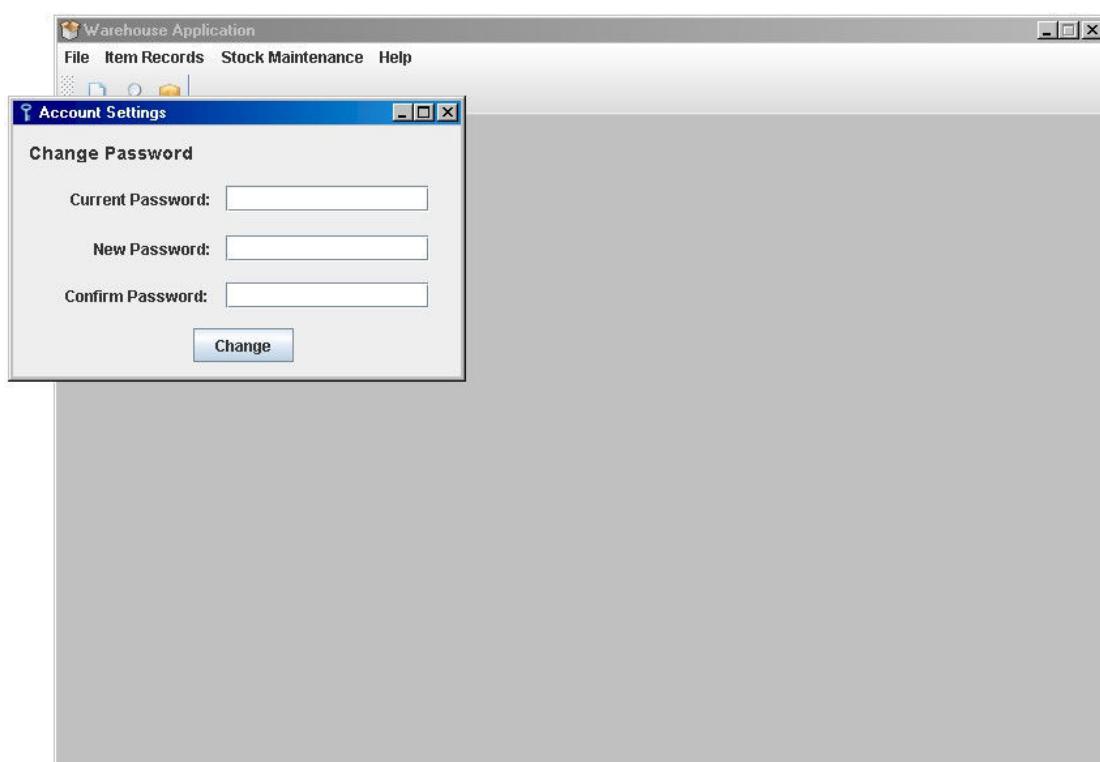
## LOADING THE ACCOUNT SETTINGS SCREEN

Clicking File > Account Settings



*Result*

Account Settings screen loads



## CHANGE USER'S PASSWORD

### CORRECT INPUT

1. Change the password of user:

Username: entryGuy  
Password: password [i.e. 'current password']

To:

Password: newPassword



Input:

Current Password: password  
New Password: newPassword  
Confirm Password: newPassword

Users file BEFORE password change

The screenshot shows a Notepad window titled 'Users.txt - Notepad'. The menu bar includes File, Edit, Format, View, and Help. The main content area displays the following text:

```
admin|root|System Administrator|1|1|1|1|
entryGuy|password|Entry Guy|0|1|0|1|
exitGuy|1234|Exit Guy|0|0|1|1|
disabled|1234|New Name|0|0|1|1|
nonAdmin|xyz|Non Admin|0|1|1|1|
```

The 'Change' button is pressed

*Result*

User is informed that the password change operation has been successful



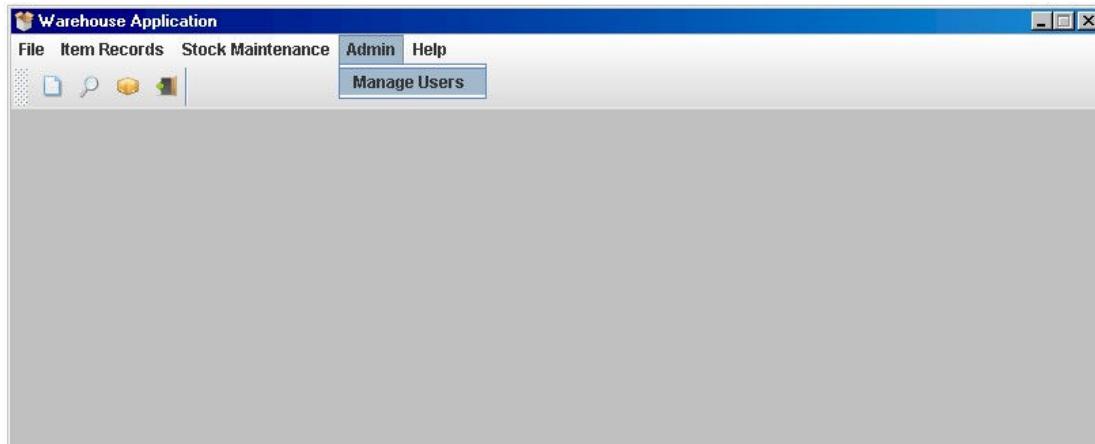
Users file AFTER password change: user's password successfully changed

A screenshot of a Windows Notepad window titled "Users.txt - Notepad". The window shows the following text content:

```
admin|root|System Administrator|1|1|1|1|
entryGuy|newPassword|Entry Guy|0|1|0|1|
exitGuy|1234|Exit Guy|0|0|1|1|
disabled|1234|New Name|0|0|1|1|
nonAdmin|xyz|Non Admin|0|1|1|1|
```

## LOADING THE MANAGE USERS SCREEN

Clicking Admin > Manage Users from the Menu Bar



*Result:*

Manage Users screen opens with a table of all the users

A screenshot of the "Manage Users" screen within the "Warehouse Application". The window title is "Manage Users". The menu bar at the top includes "File", "Item Records", "Stock Maintenance", "Admin", and "Help". Below the menu bar is a toolbar with four icons. The main area contains a table with the following data:

Username	Password	Full Name	Admin	Entry	Exit	Enabled
admin	root	System Administ...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
entryGuy	password	Entry Guy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
exitGuy	1234	Exit Guy	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
disabled	abcd	Disabled User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
nonAdmin	xyz	Non Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

At the bottom of the screen are three buttons: "New User", "Delete", and "Edit".

## SELECTING A USER

### CORRECT INPUT

1. Clicking one row

*Result:* The row is selected

The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", "Admin", and "Help". Below the menu is a toolbar with icons for file operations. The main area is titled "Manage Users" and contains a table with the following data:

Username	Password	Full Name	Admin	Entry	Exit	Enabled
admin	root	System Administ...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
entryGuy	password	Entry Guy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
exitGuy	1234	Exit Guy	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
disabled	abcd	Disabled User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
nonAdmin	xyz	Non Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

At the bottom of the window are three buttons: "New User", "Delete", and "Edit".

### INCORRECT INPUT

1. Click-and-drag to attempt to select multiple rows

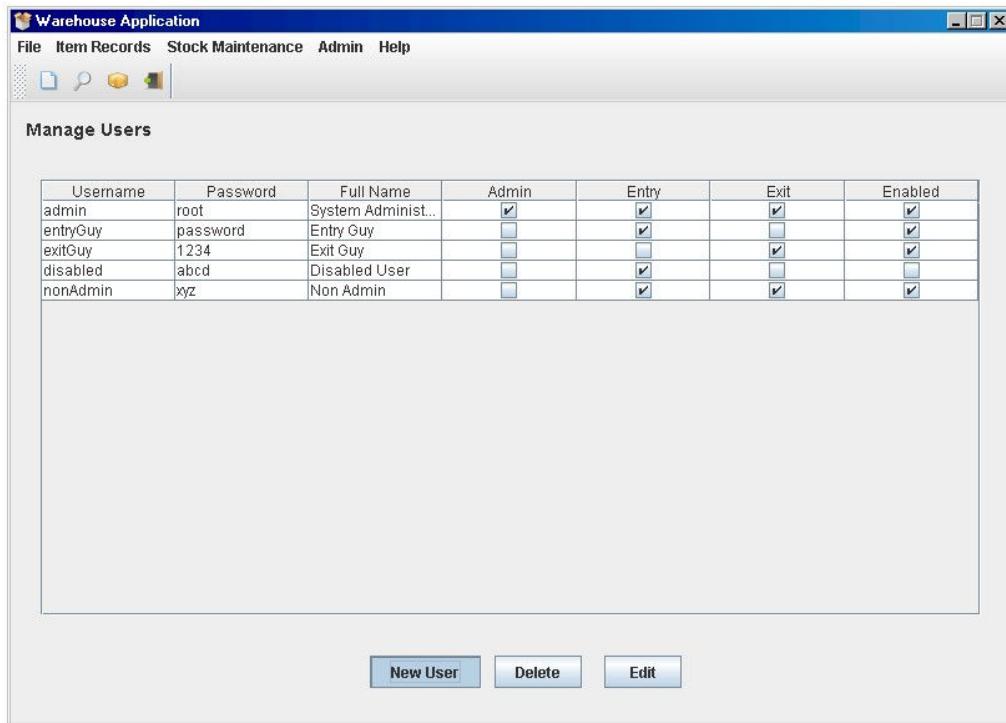
*Result*

Multiple row selections not allowed. Only one row selected

The screenshot shows the same "Warehouse Application" window and "Manage Users" table as the previous image. In this version, multiple rows are highlighted with blue selection bars on the left side of the table. However, only the first row (admin) has its checkboxes checked, indicating that only one row was successfully selected.

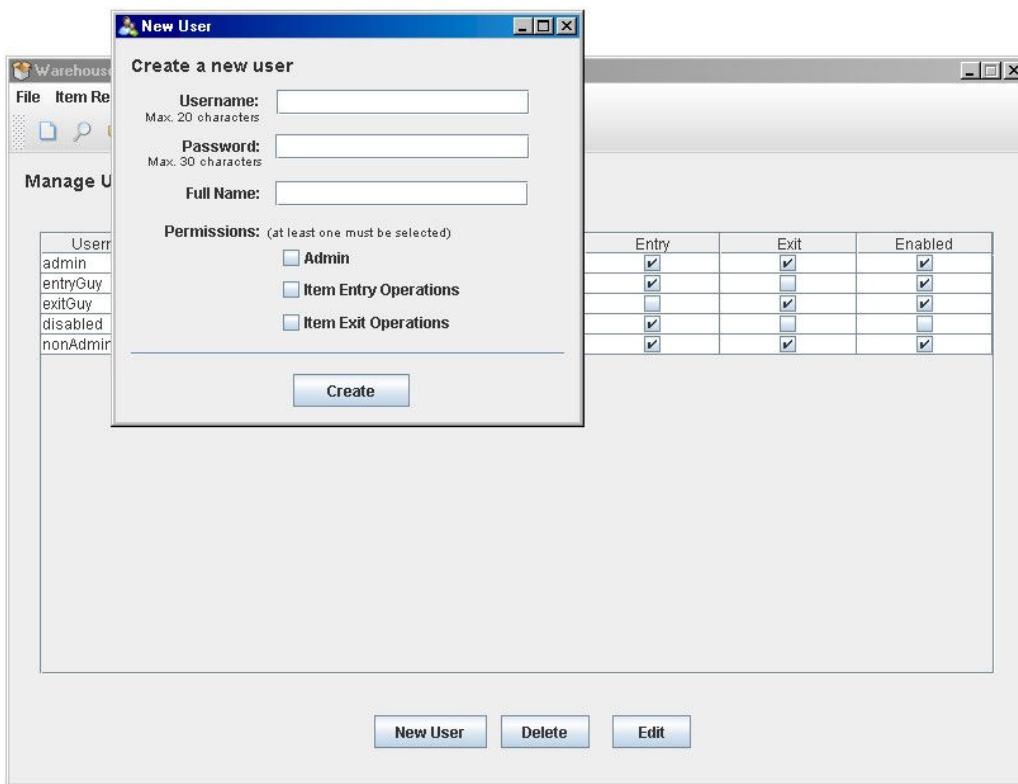
## LOADING THE NEW USER SCREEN

Clicking the 'New User' button



*Result*

New User screen opens



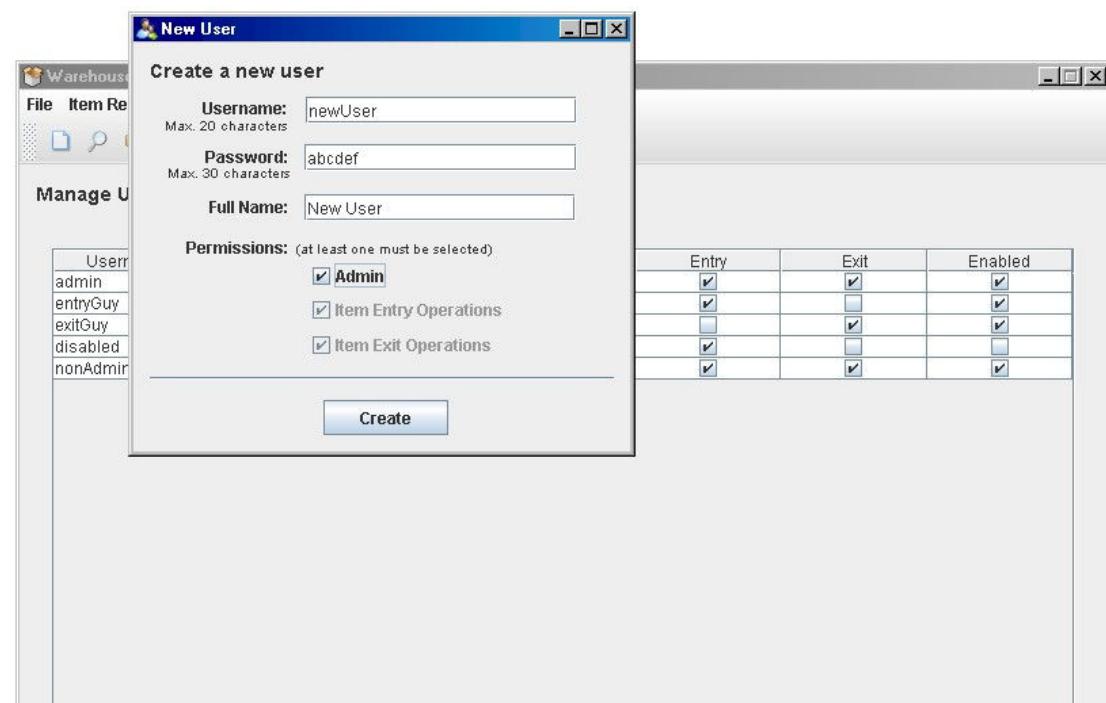
## CREATING A NEW USER

### CORRECT INPUT

1. Username max 20 characters, password max 30 characters, user's full name, at least one permission selected

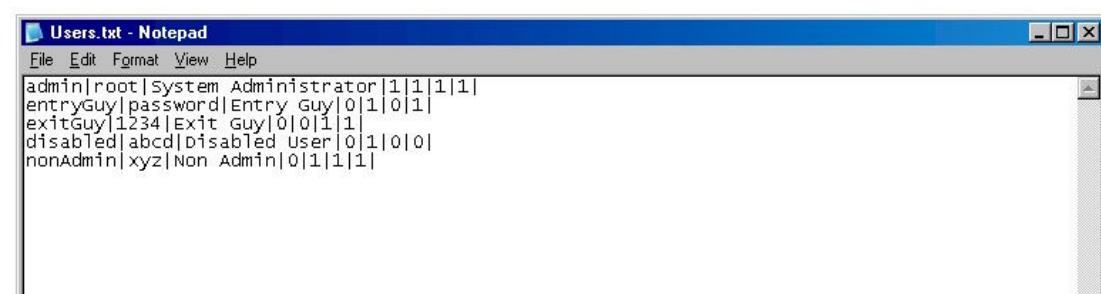
*Test*

Username: newUser  
Password: abcdef  
Full Name: New User  
Permissions: Admin Selected



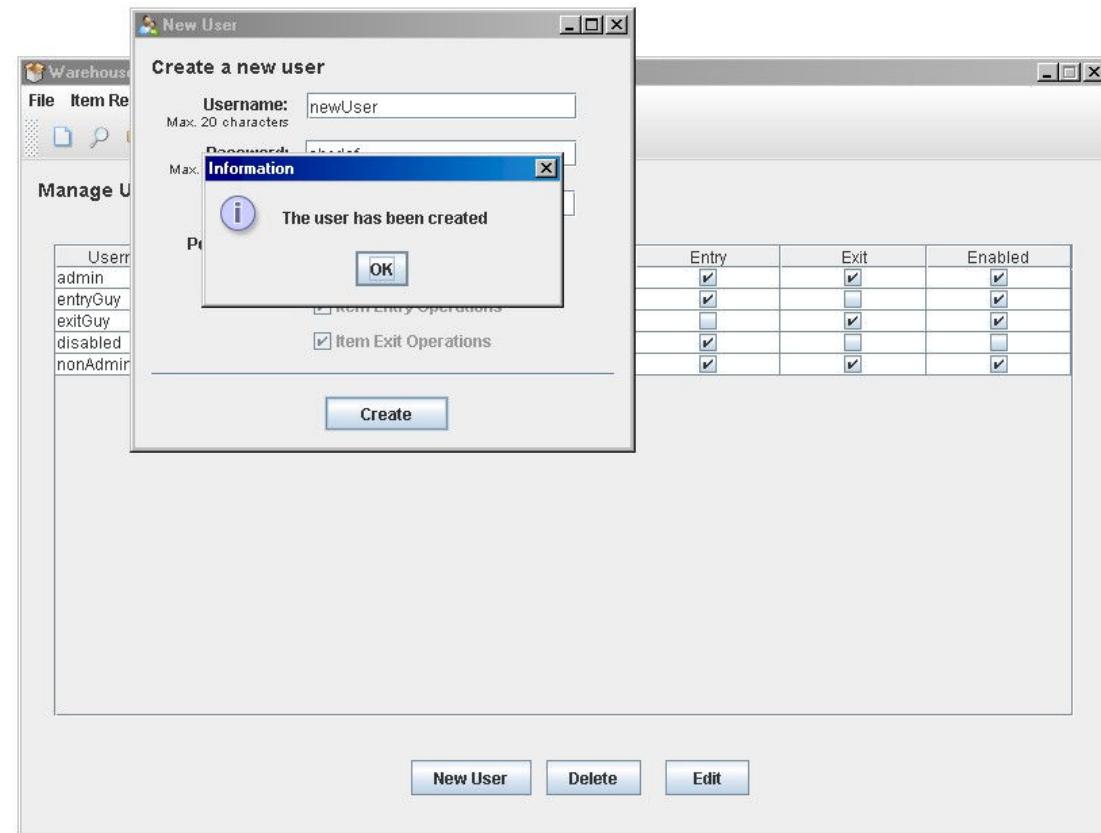
Note: when the 'Admin' checkbox is selected, the other permissions automatically select and become disabled (an admin user has both entry and exit permissions).

*Users File BEFORE user creation*

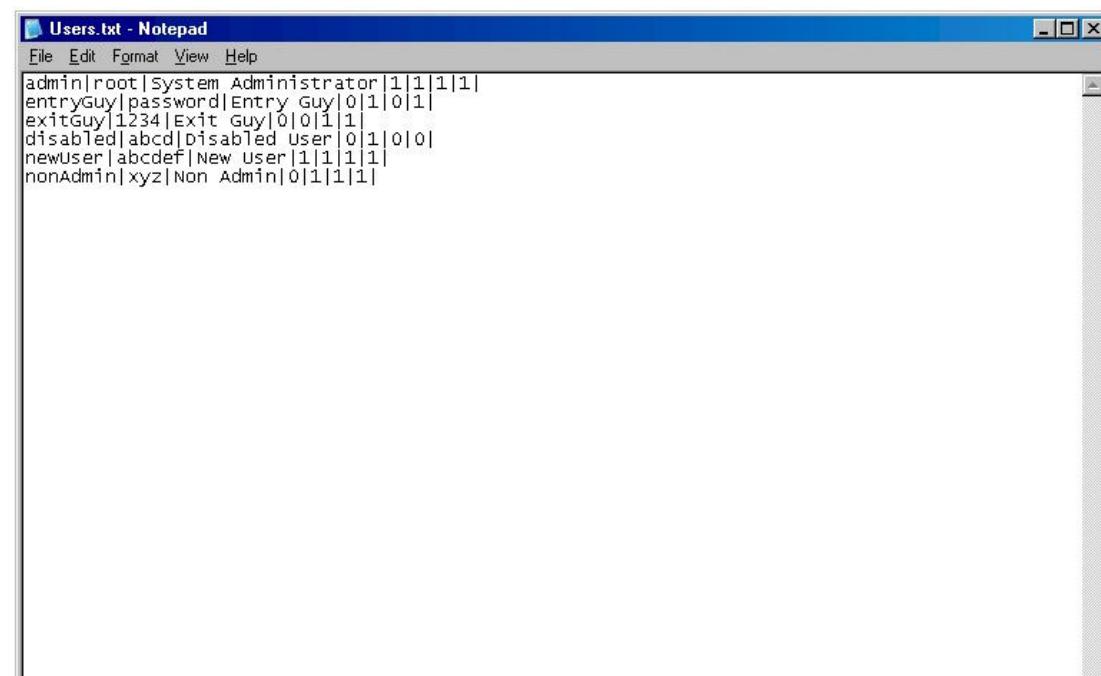


## *Result*

User is informed that the user was created



*Users file AFTER user creation*



The user was created, with correct fields and appropriate permissions. This new user is displayed in the table

Username	Password	Full Name	Admin	Entry	Exit	Enabled
admin	root	System Administ...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
entryGuy	password	Entry Guy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
exitGuy	1234	Exit Guy	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
disabled	abcd	Disabled User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
newUser	abcdef	New User	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nonAdmin	xyz	Non Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Buttons at the bottom: New User, Delete, Edit.

## INCORRECT INPUT

1. One or more fields left blank or no permissions selected

*Test:*

Username: wrongInput  
Password: [blank]  
Full Name: Wrong Input  
Permissions: [none selected]

New User

Create a new user

Username: wrongInput  
Max. 20 characters

Password: [blank]  
Max. 30 characters

Full Name: Wrong Input

Permissions: (at least one must be selected)

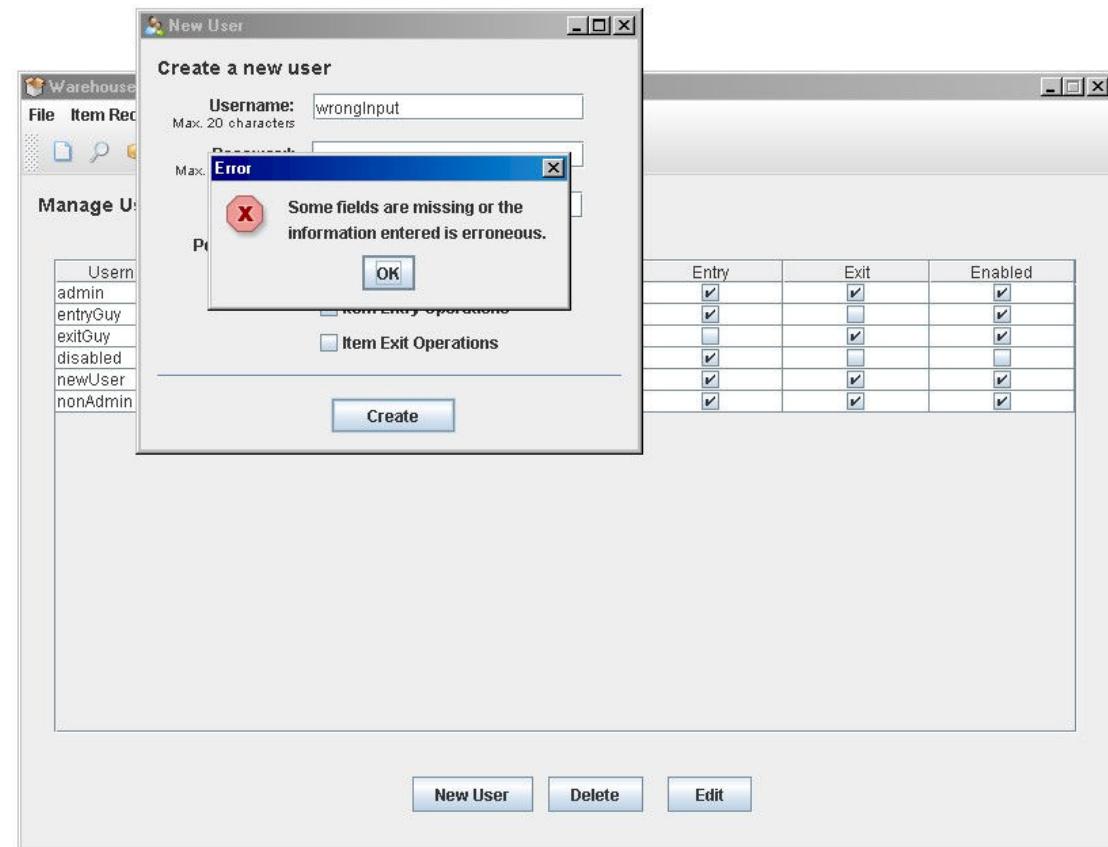
	Entry	Exit	Enabled
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Create

The create button is clicked

*Result:*

The user is informed that some fields are missing or incorrect



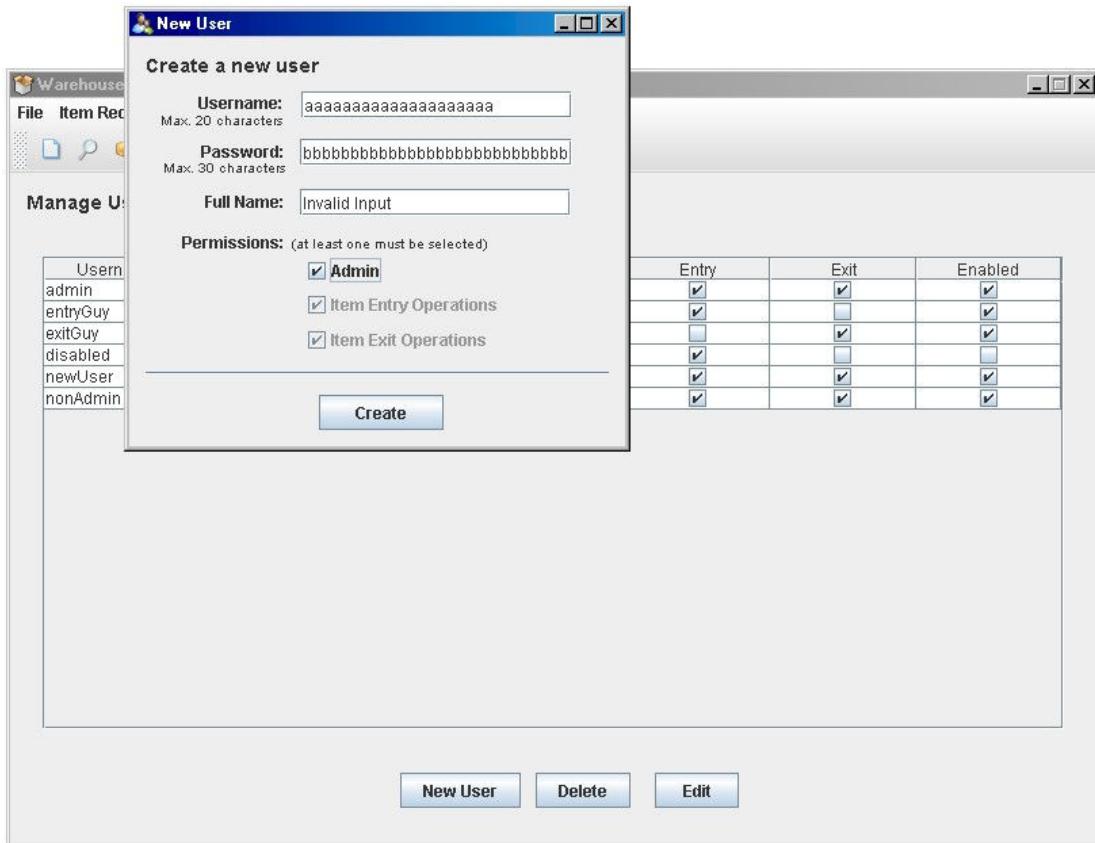
**2. Username longer than 20 characters and password longer than 30 characters**

*Test:*

Username:aaaaaaaaaaaaaaaaaaaaaaaaaaaa [25 characters]  
Password:bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb [35 characters]  
FullName: Invalid Input  
Permissions: Admin

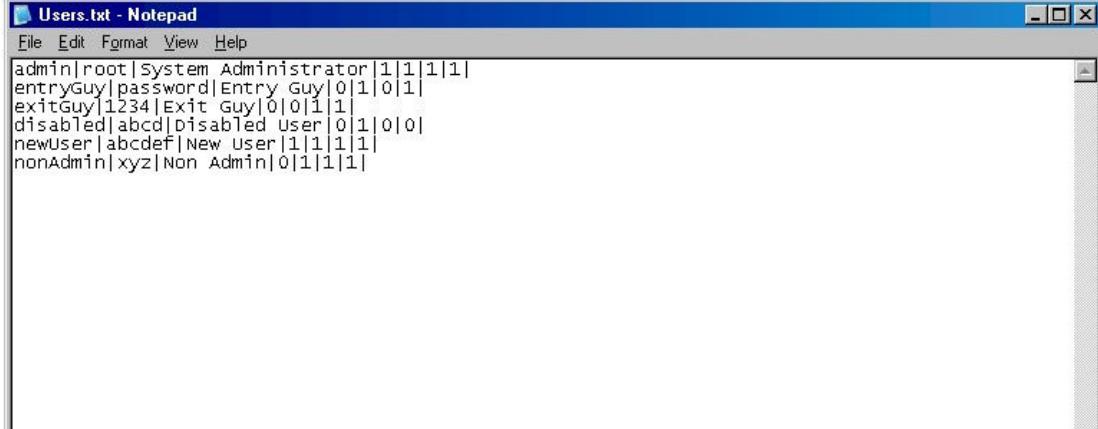
*Result:*

Textfields do not allow for more than 20 and 30 characters respectively, and truncate user input.



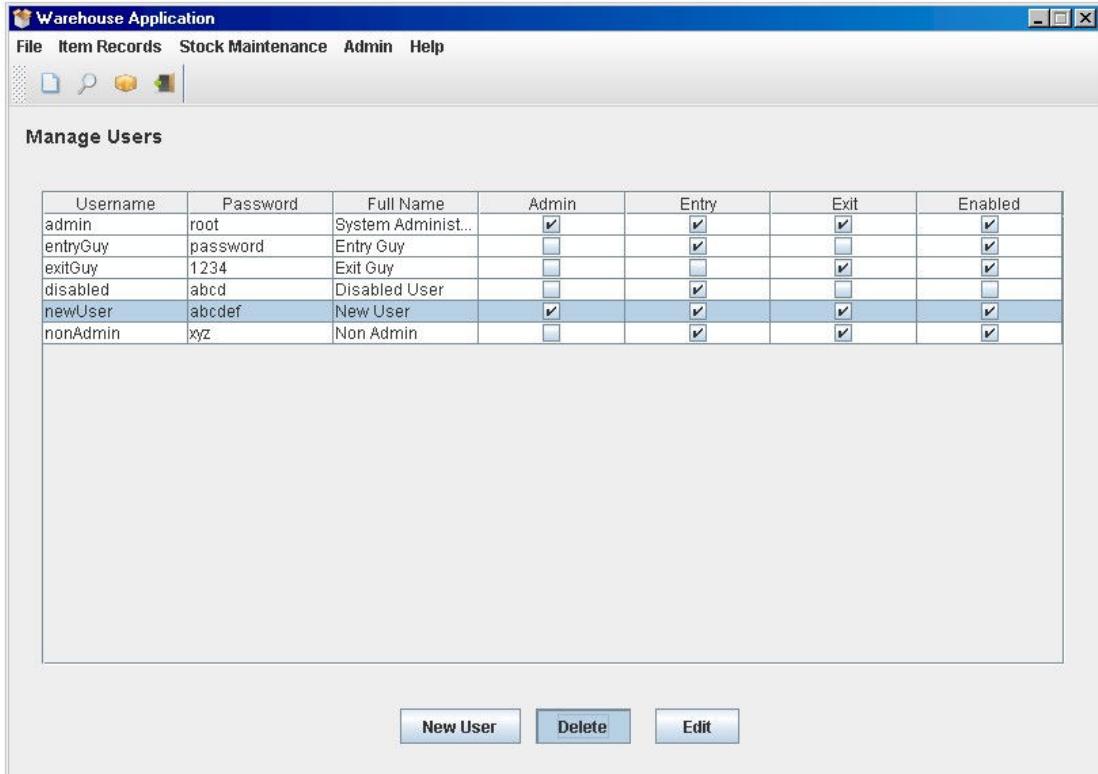
## DELETE A USER

*Users file before deletion*



```
admin|root|System Administrator|1|1|1|1|  
entryGuy|password|Entry Guy|0|1|0|1|  
exitGuy|1234|Exit Guy|0|0|1|1|  
disabled|abcd|Disabled User|0|1|0|0|  
newUser|abcdef|New User|1|1|1|1|  
nonAdmin|xyz|Non Admin|0|1|1|1|
```

To delete a user, the row containing the user with the username 'newUser' is selected and the 'Delete' button is pressed

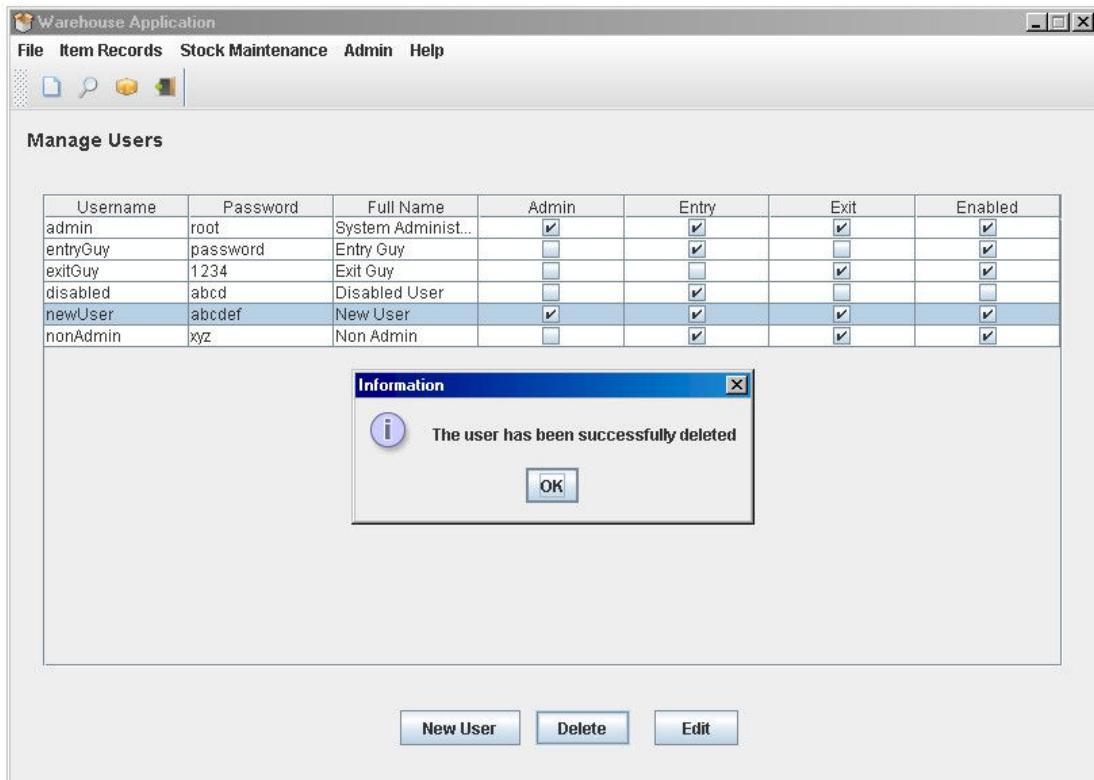


The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes File, Item Records, Stock Maintenance, Admin, and Help. Below the menu is a toolbar with icons for New User, Delete, and Edit. The main area is titled "Manage Users" and contains a grid table with the following data:

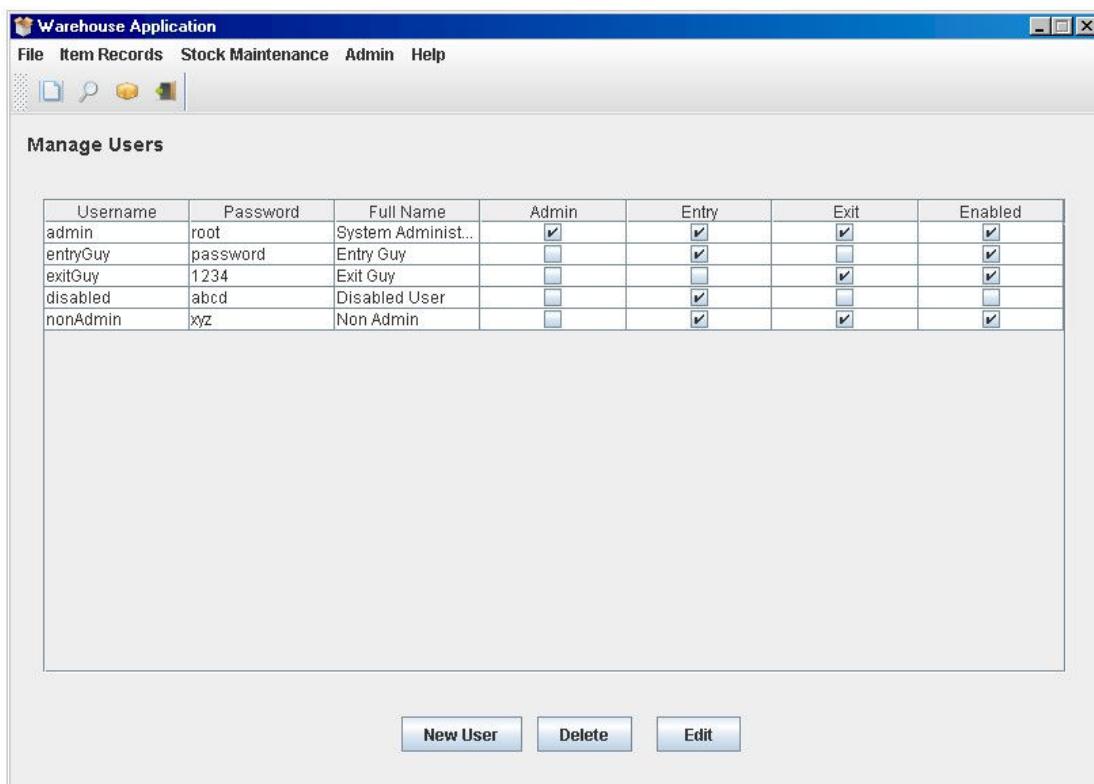
Username	Password	Full Name	Admin	Entry	Exit	Enabled
admin	root	System Administ...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
entryGuy	password	Entry Guy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
exitGuy	1234	Exit Guy	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
disabled	abcd	Disabled User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
newUser	abcdef	New User	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nonAdmin	xyz	Non Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

*Result*

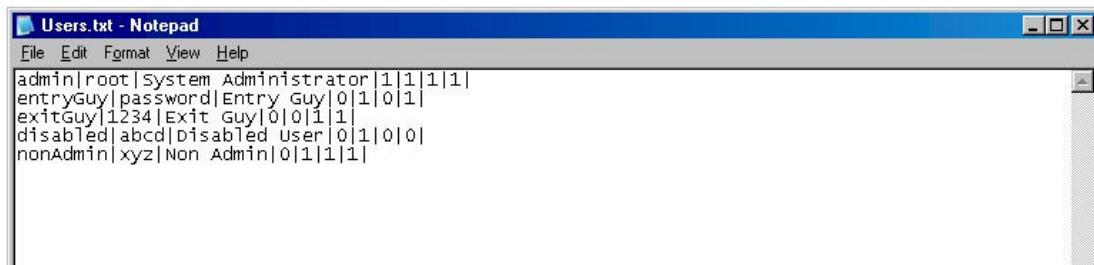
The user is informed that the user has been deleted



The table updates without the deleted user



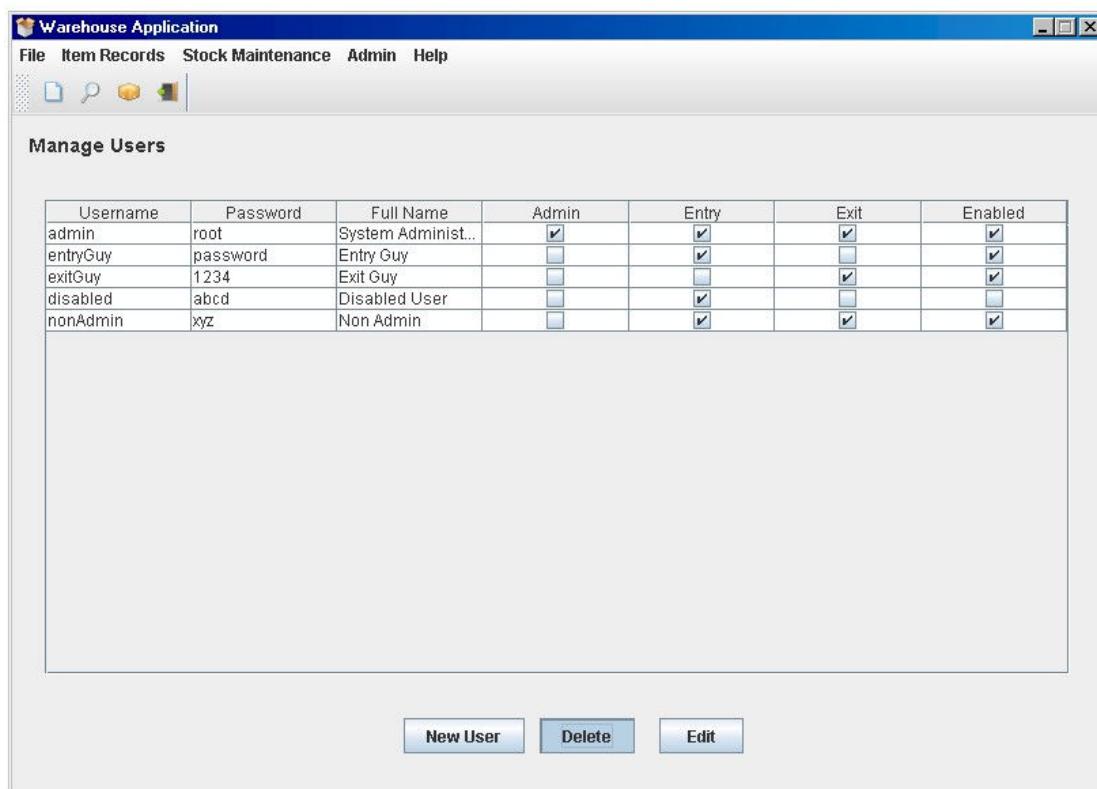
Users file after deletion (user has been deleted)



```
admin|root|System Administrator|1|1|1|1|  
entryGuy|password|Entry Guy|0|1|0|1|  
exitGuy|1234|Exit Guy|0|0|1|1|  
disabled|abcd|Disabled User|0|1|0|0|  
nonAdmin|xyz|Non Admin|0|1|1|1|
```

### **INCORRECT INPUT**

1. Pressing the 'Delete' button without selecting any user



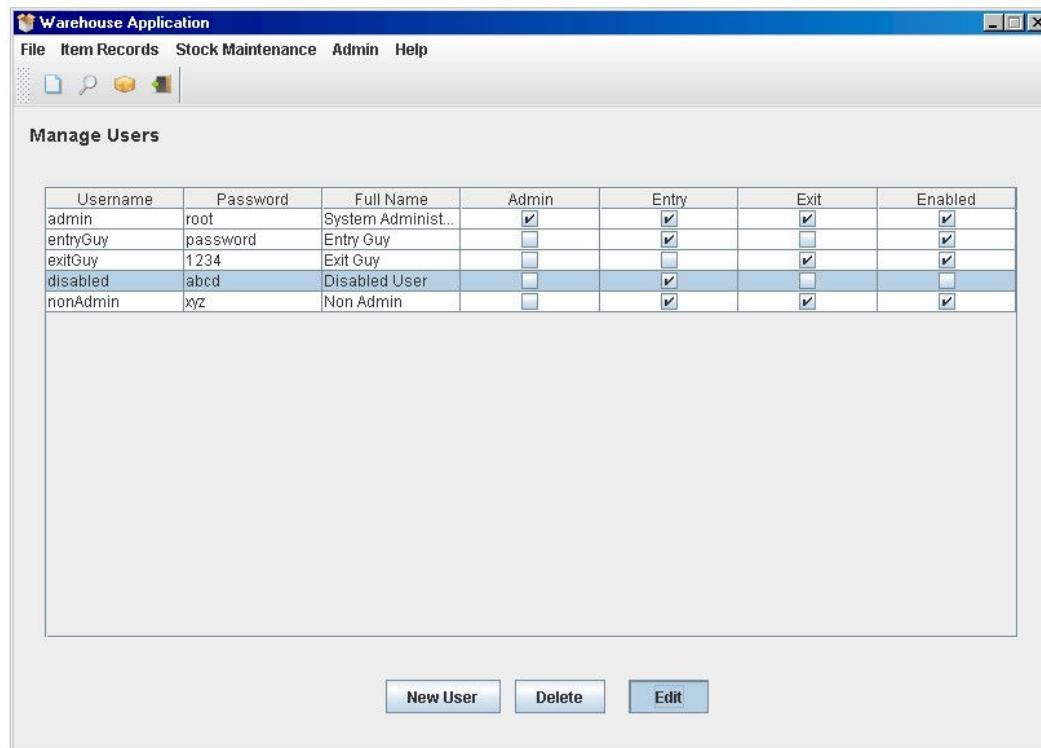
*Result:*

Nothing happens (error handling successful)

## LOADING THE EDIT USER SCREEN

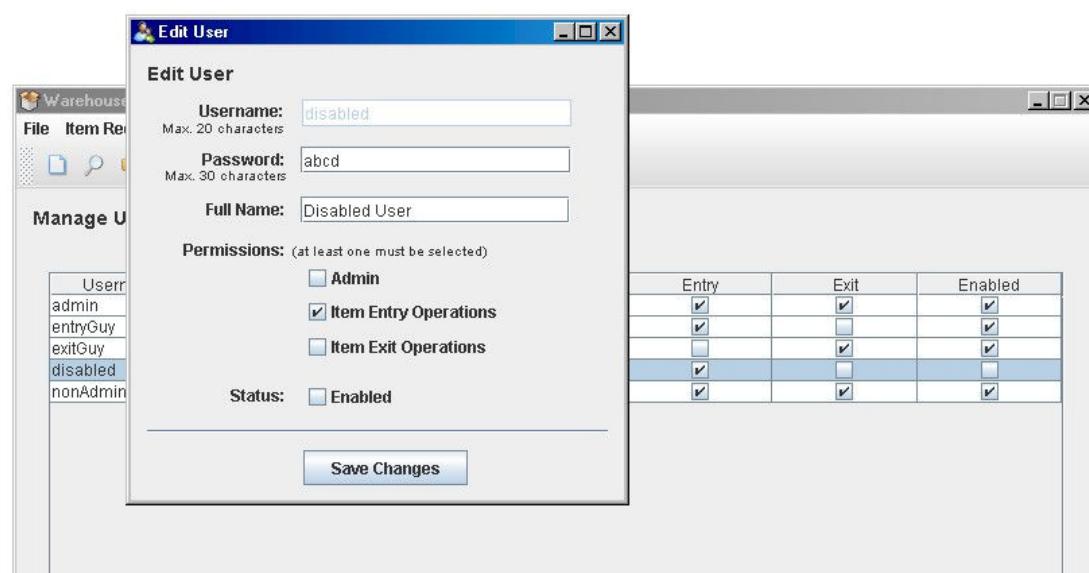
To load the edit user screen, select the row containing the and click the 'Edit' button

Edit user with username 'disabled' by clicking the row containing the user and clicking 'edit'



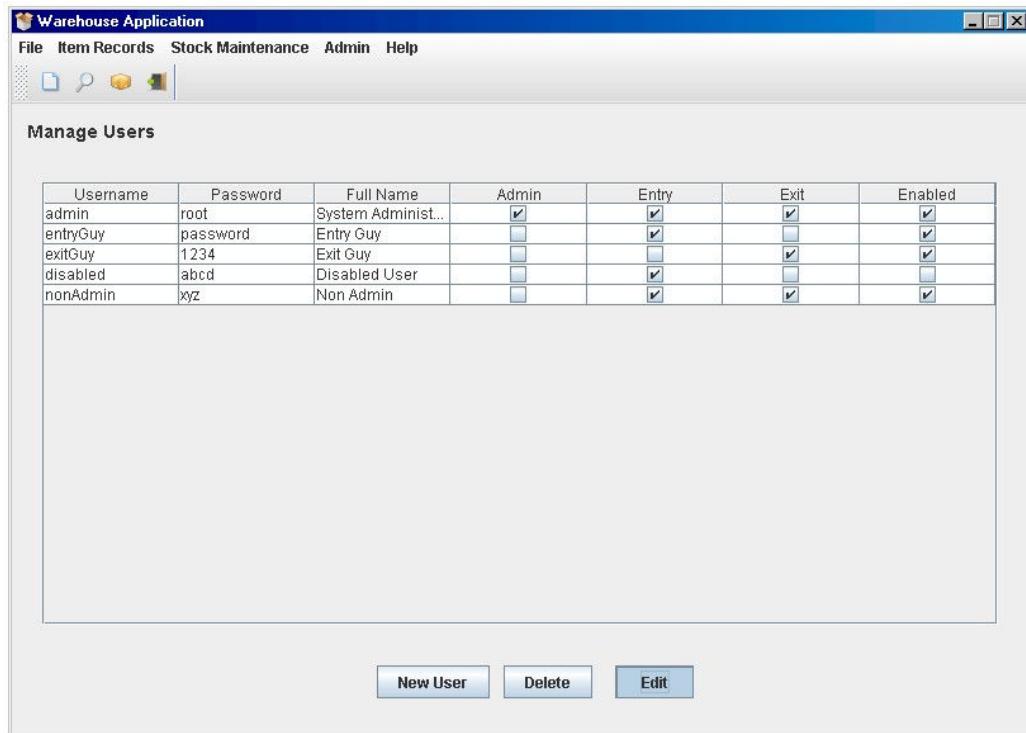
### Result

Edit user screen opens with correct information of that user



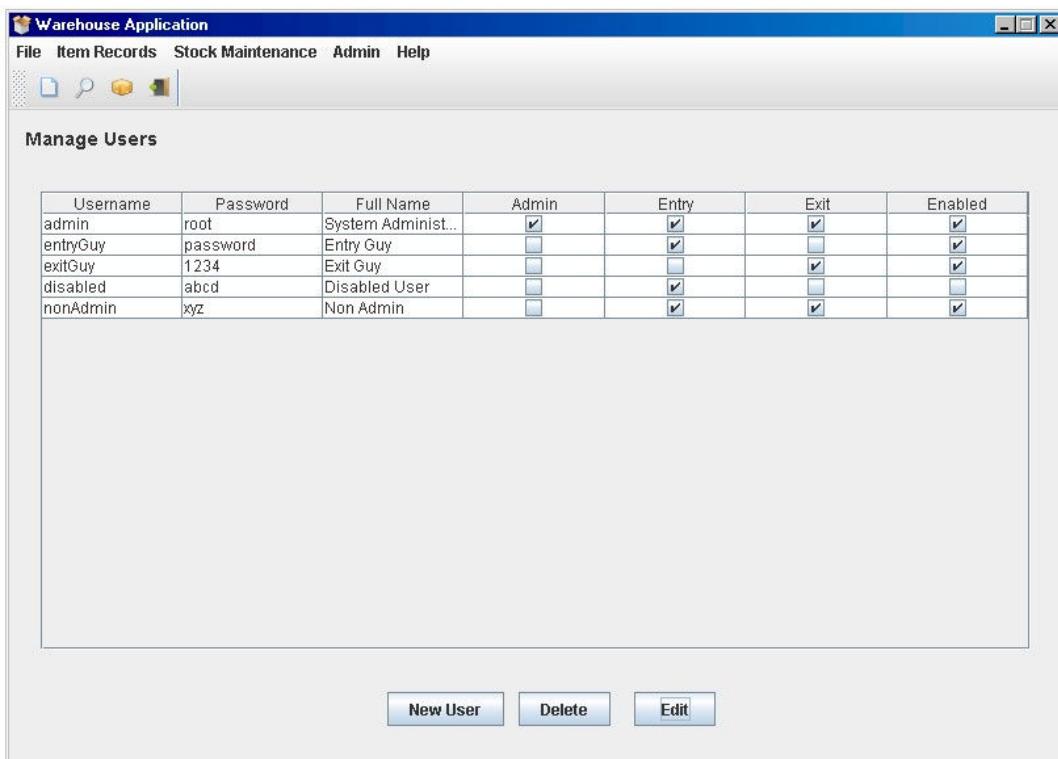
## INCORRECT INPUT

Clicking the edit button with no user selected



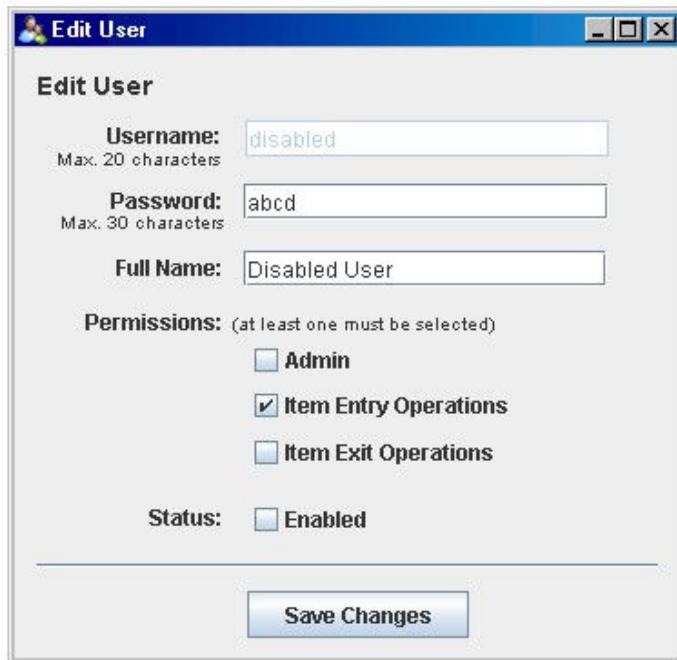
### *Result*

Nothing happens (error handling successful)



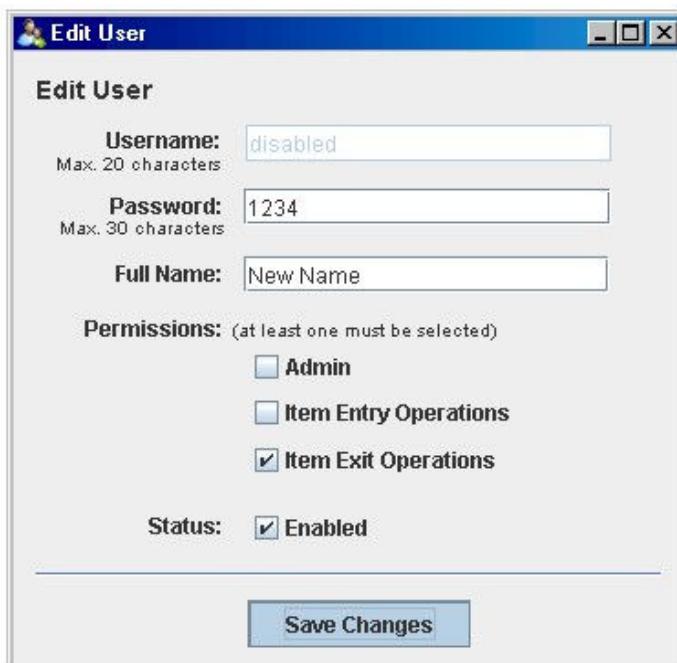
## EDITING A USER

Edit the user with username 'disabled'

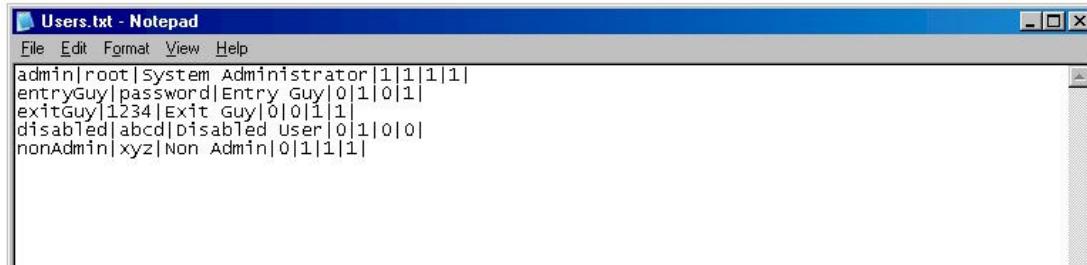


Edited to:

Password: 1234  
Full Name: New Name  
Permissions: Item Exit  
Status: Enabled



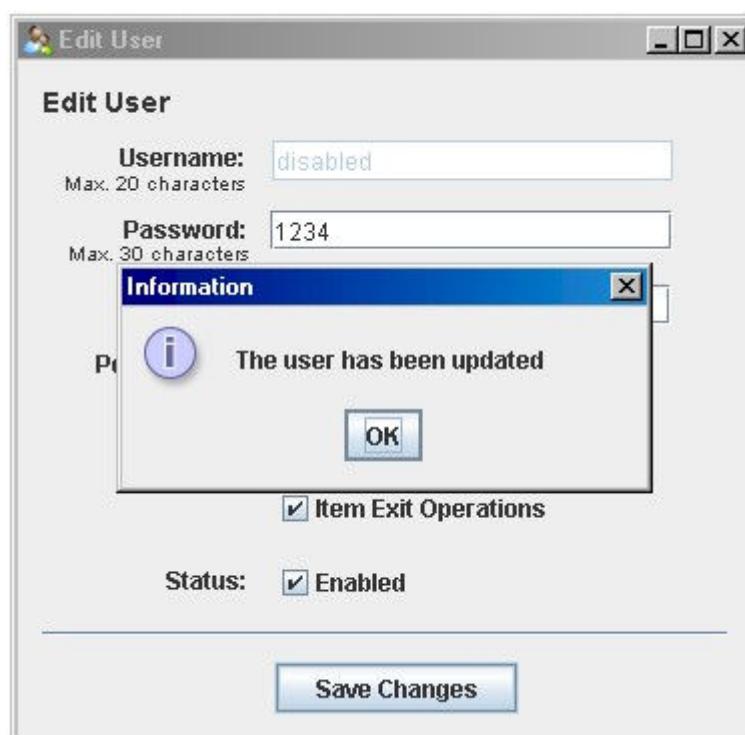
## Users File BEFORE editing



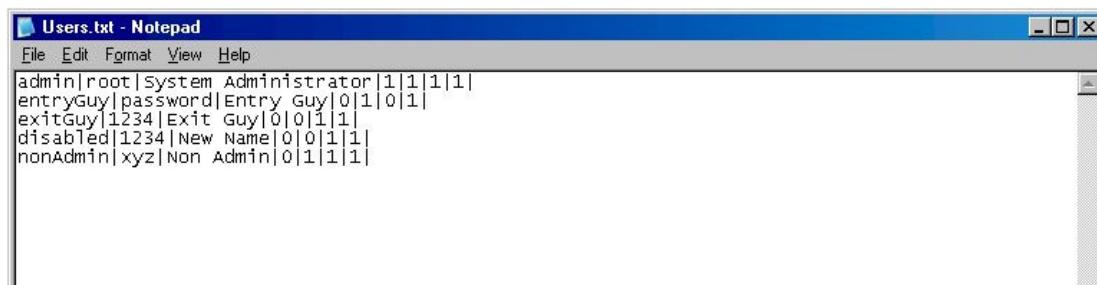
The ‘Save Changes’ button is pressed

## Result:

User is informed that the edit operation has succeeded



Users file AFTER user updated (User edited)



## INCORRECT INPUT

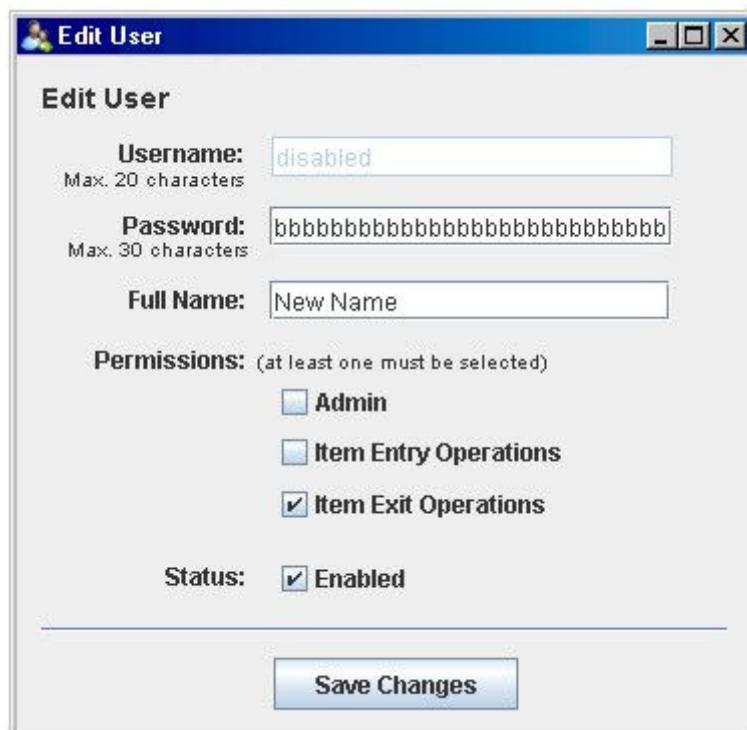
1. Password longer than 30 characters.

*Test:*

Password: bbbbbbbbbbbbbb [35 characters]  
FullName: Invalid Input  
Permissions: Admin

*Result*

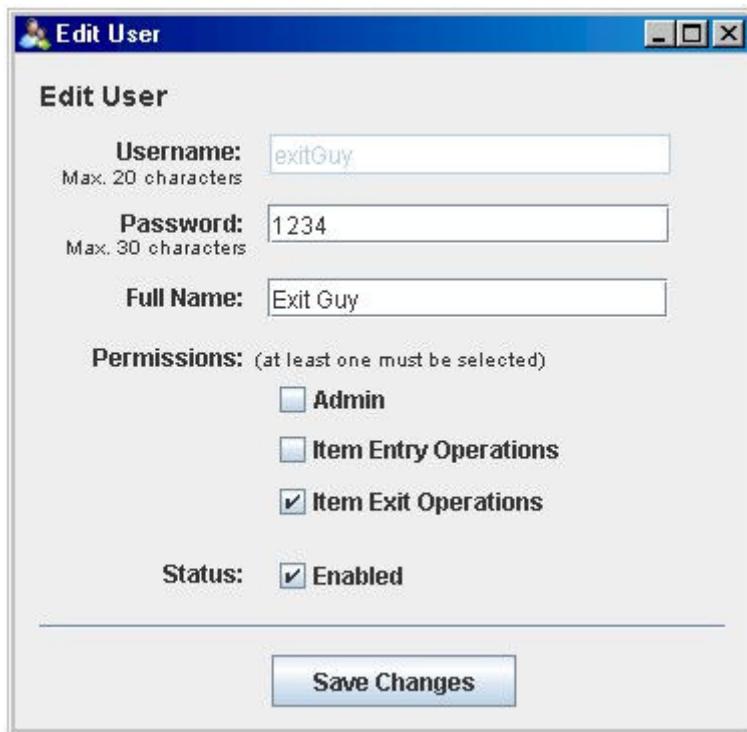
Password truncated to maximum length (30 characters)



2. Some fields missing and/or no permissions set

*Test*

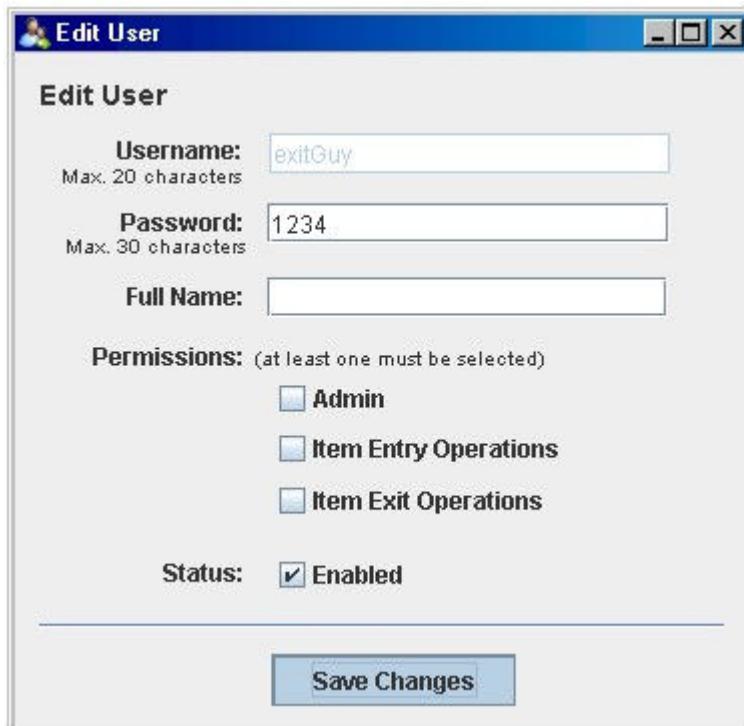
Change this user's information from



To

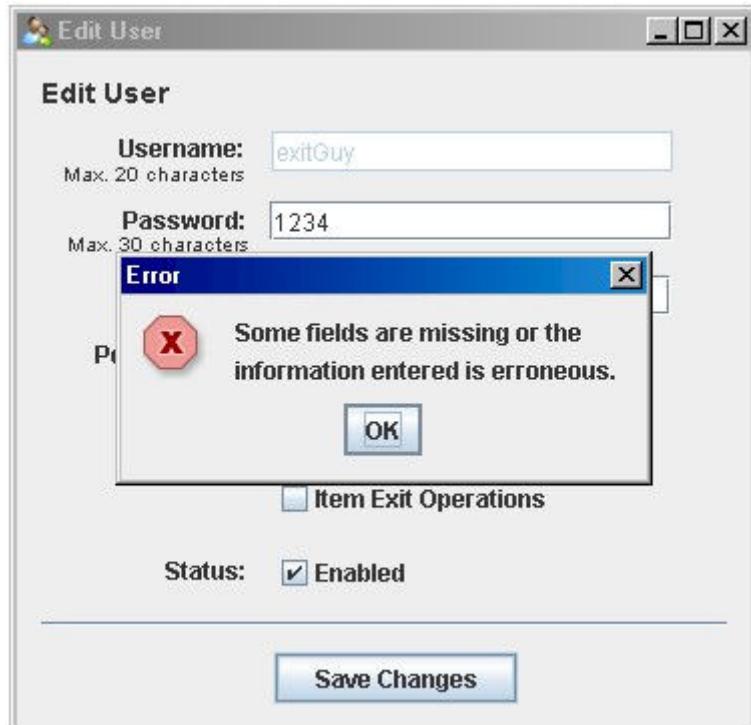
Password: 1234  
Full Name: [left blank]  
Permissions: [none]  
Status: Enabled

The 'Save Changes' button is pressed



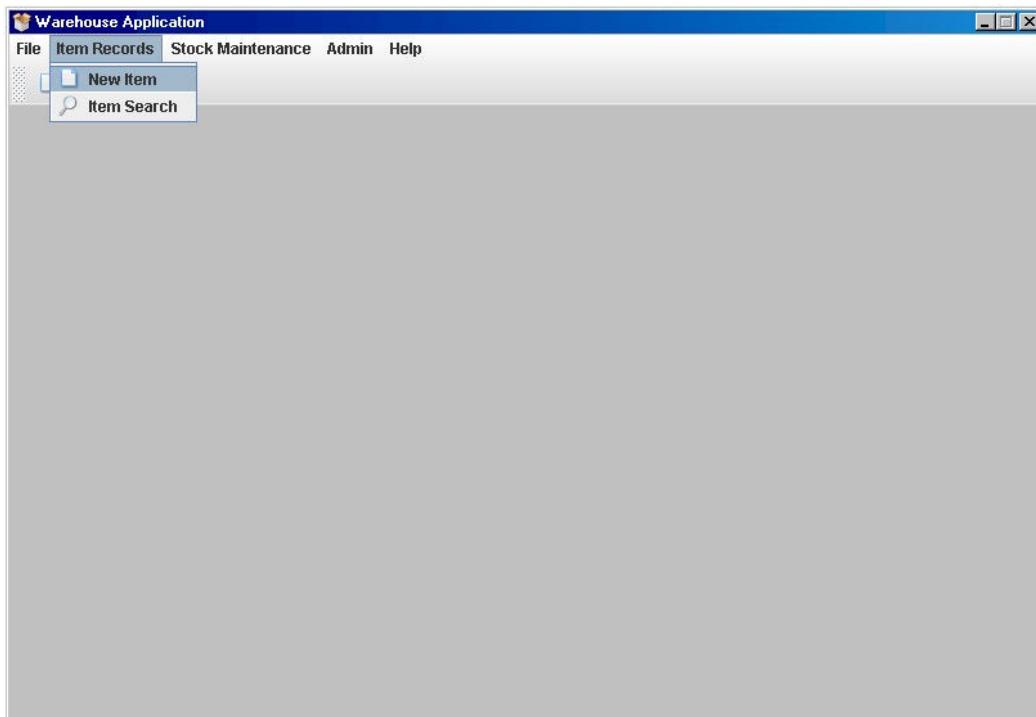
*Result*

The edit operation does not proceed. The user is informed.



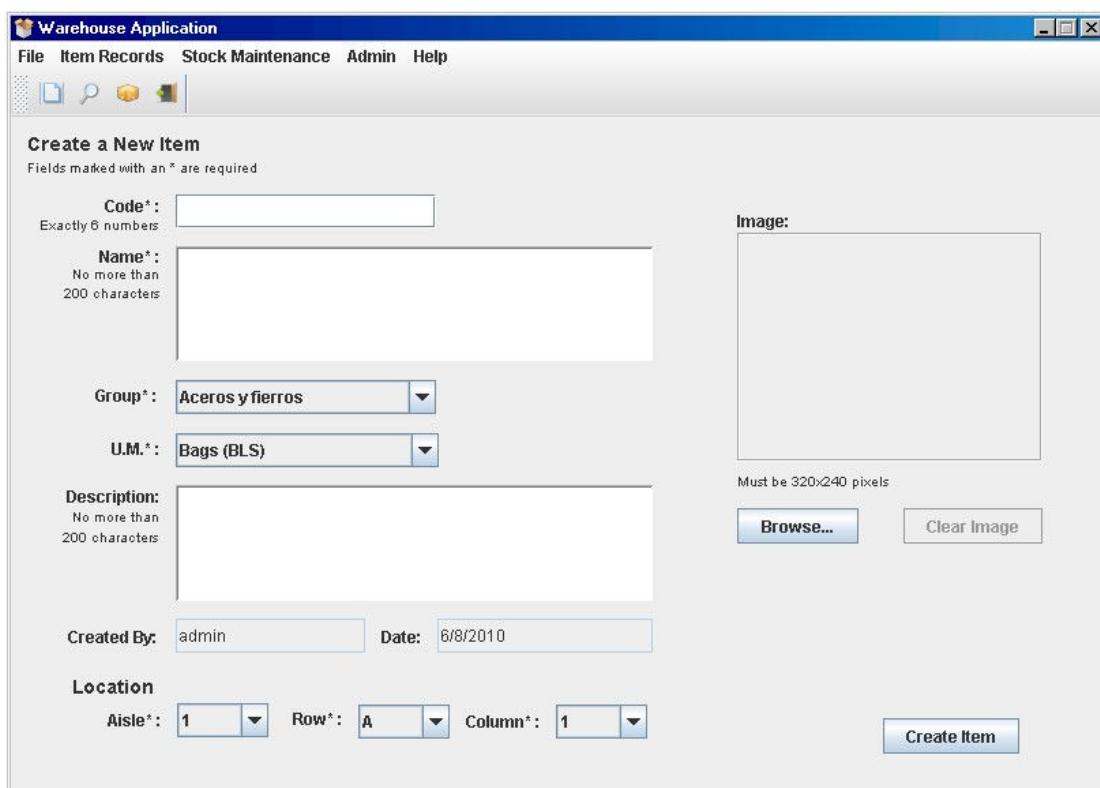
## LOADING THE CREATE A NEW ITEM SCREEN

Clicking Item Records > New Item from the Menu Bar



### Result

Create New Item screen loads

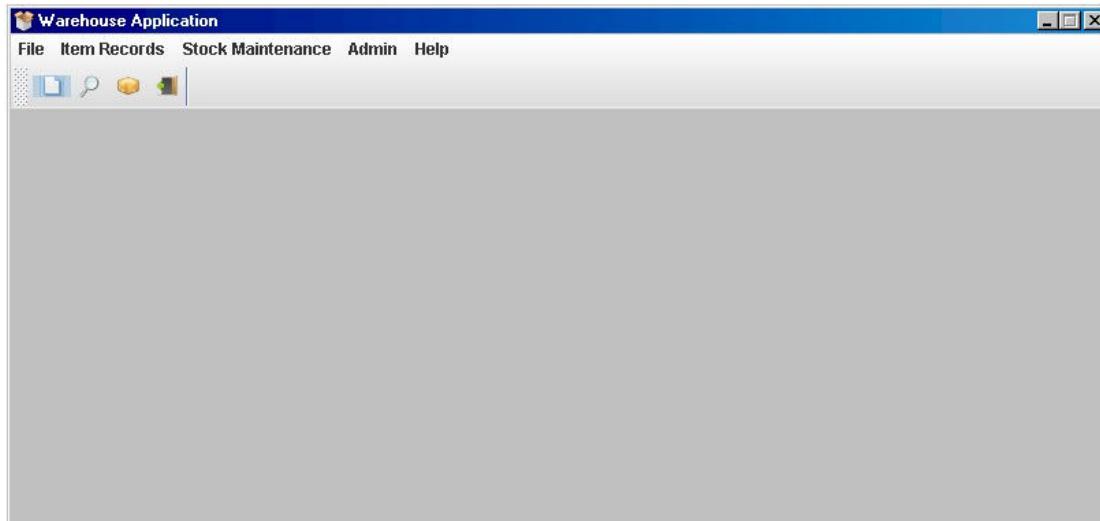


The screenshot shows the "Create a New Item" form. The form has the following fields:

- Code\*:** Exactly 6 numbers (input field)
- Name\*:** No more than 200 characters (input field)
- Group\*:** Aceros y fierros (dropdown menu)
- U.M.\*:** Bags (BLS) (dropdown menu)
- Description:** No more than 200 characters (input field)
- Created By:** admin (input field)
- Date:** 6/8/2010 (input field)
- Location**:  
Aisle\*: 1 (dropdown menu)   Row\*: A (dropdown menu)   Column\*: 1 (dropdown menu)
- Image:** Must be 320x240 pixels (input field for image file, with Browse... and Clear Image buttons)
- Create Item** (button at the bottom right)

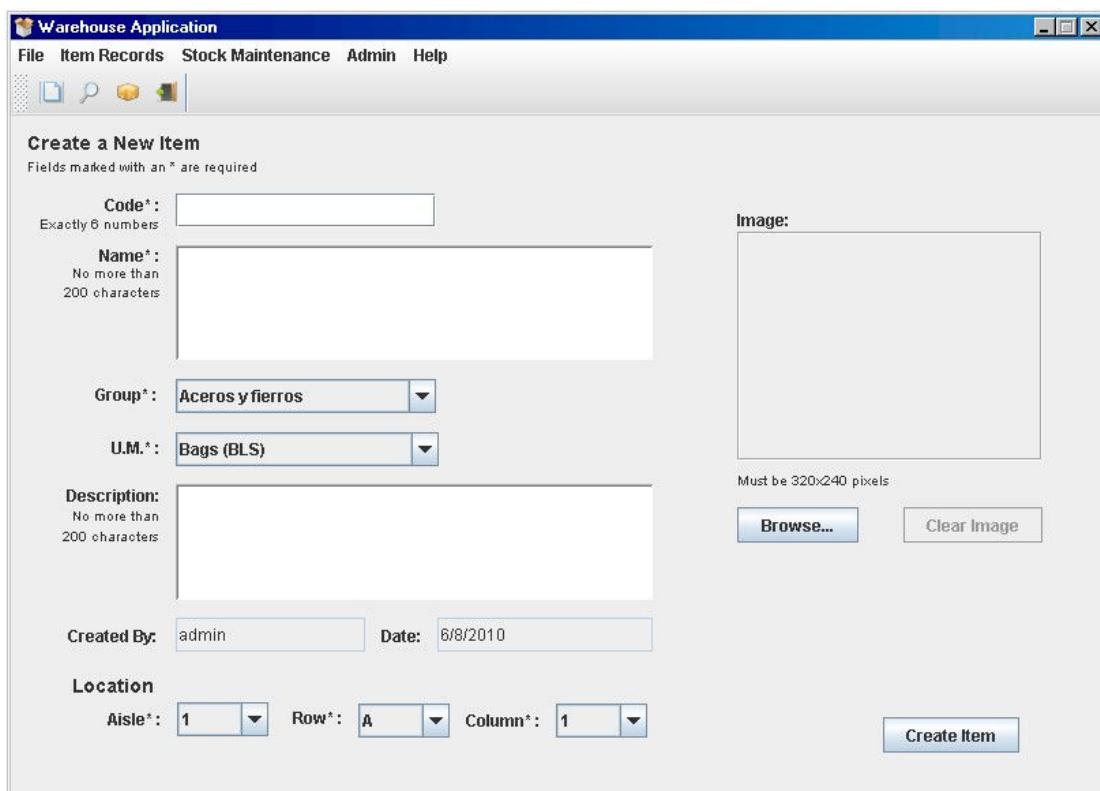
OR

Clicking the New Item button in the Toolbar



*Result*

Create New Item screen loads



**Create a New Item**  
Fields marked with an \* are required

**Code\***:  Exactly 6 numbers

**Name\***:  No more than 200 characters

**Group\***: Aceros y fierros

**U.M.\***: Bags (BLS)

**Description**:  
No more than 200 characters

**Created By**: admin      **Date**: 8/8/2010

**Location**

Aisle\*: 1    Row\*: A    Column\*: 1

Image:  
Must be 320x240 pixels

Browse...    Clear Image

Create Item

Note that the user that logged in is assigned as the creator of this item and the current date is set.

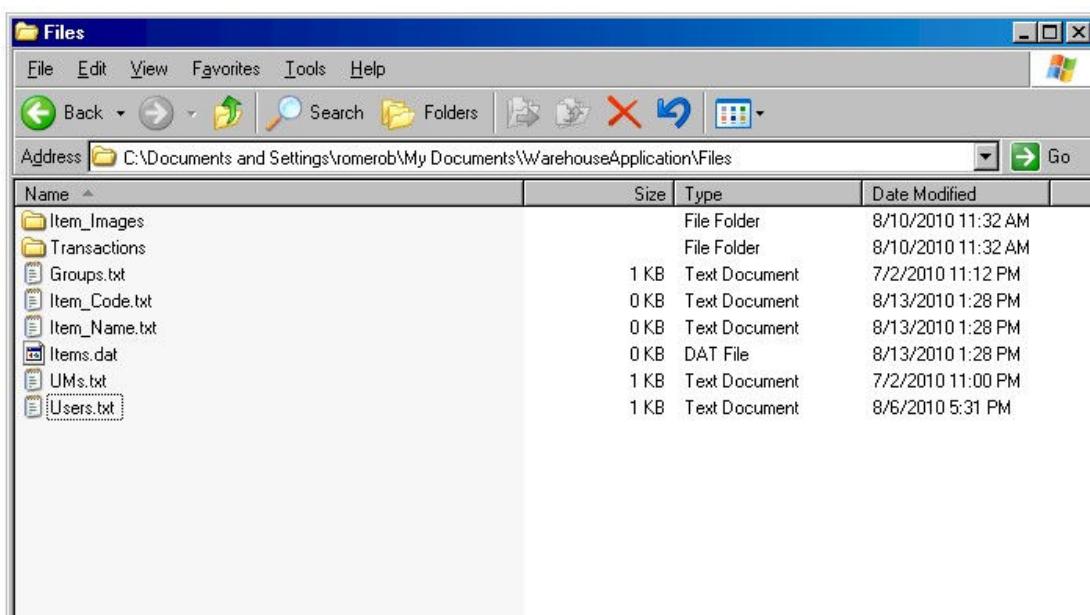
## CREATING A NEW ITEM

The various components of the item creation process will be tested for different data sets

### CORRECT INPUT

#### 1. Creating an item with no image

No items are in the system. The Items random access file is NOT initialised (size is 0 KB)



The following item will be created

Code: 398749  
Name: Bosch Drill with changeable drill heads  
Group: Herramientas  
U.M.: Unit (U)  
Description: Red body, up to 900 MW, 220 AC input  
Location:  
    Aisle: 52  
    Row: B  
    Column: 7

For this test, no item image will be assigned

Note that the item is assigned a creation date and a creator

The 'Create Item' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Help

Create a New Item  
Fields marked with an \* are required

**Code\***: 398749  
Exactly 6 numbers

**Name\***: Bosch Drill with changeable drill heads  
No more than 200 characters

**Group\***: Herramientas

**U.M.\***: Unit (U)

**Description:**  
No more than 200 characters  
Red body, up to 900 MW, 220 AC input

**Created By:** entryGuy    **Date:** 13/8/2010

**Location**  
**Aisle\***: 52    **Row\***: B    **Column\***: 7

**Image:**  
Must be 320x240 pixels

*Result:*

The user is informed that the item was created

**Warehouse Application**

File Item Records Stock Maintenance Help

Create a New Item  
Fields marked with an \* are required

**Code\***: 398749  
Exactly 6 numbers

**Name\***: Bosch Drill with changeable drill heads  
No more than 200 characters

**Group\***: Herramientas

**U.M.\***: Unit (U)

**Description:**  
No more than 200 characters  
Red body, up to 900 MW, 2

**Created By:** entryGuy    **Date:** 13/8/2010

**Location**  
**Aisle\***: 52    **Row\***: B    **Column\***: 7

**Image:**  
Must be 320x240 pixels

**Success**  
The item has been created.

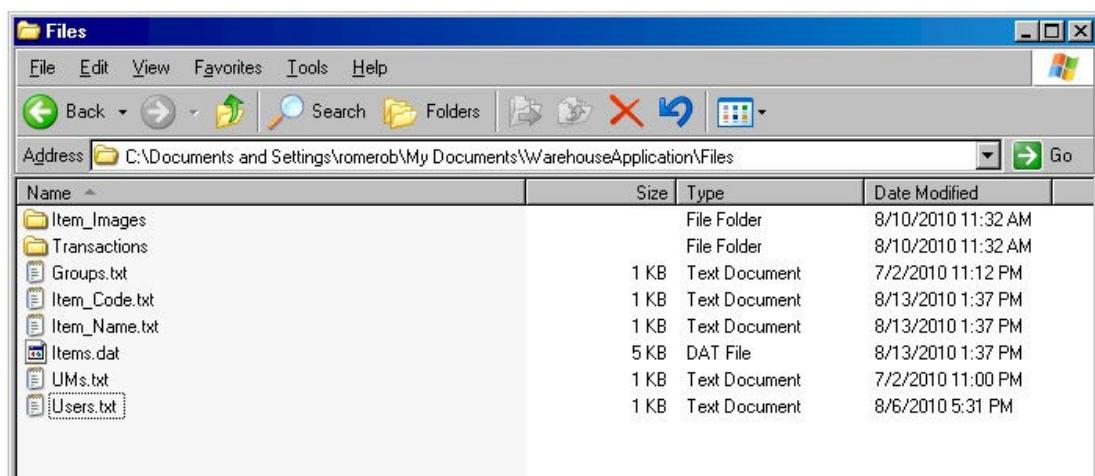
Click 'OK'

*Result*

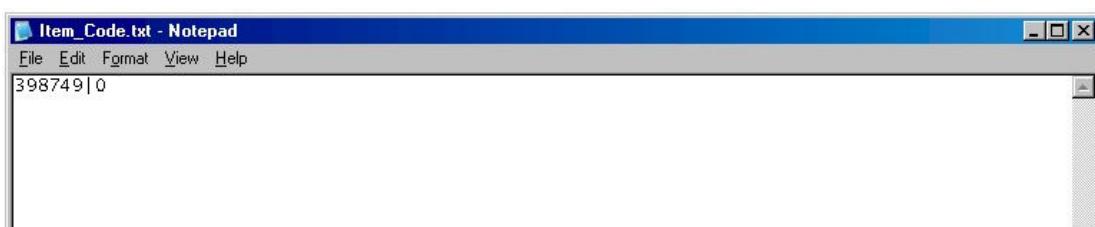
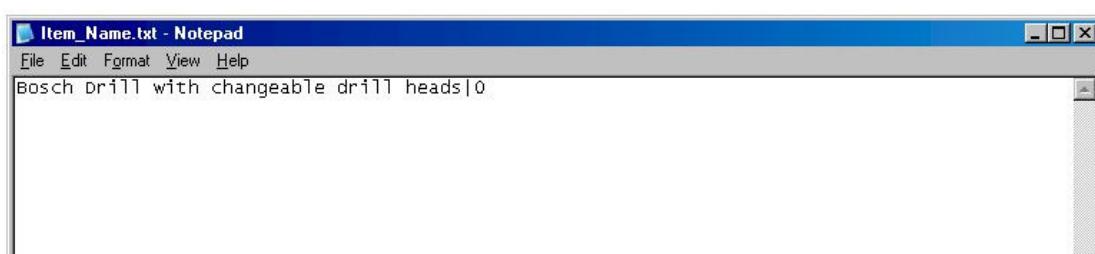
The application returns to the landing screen



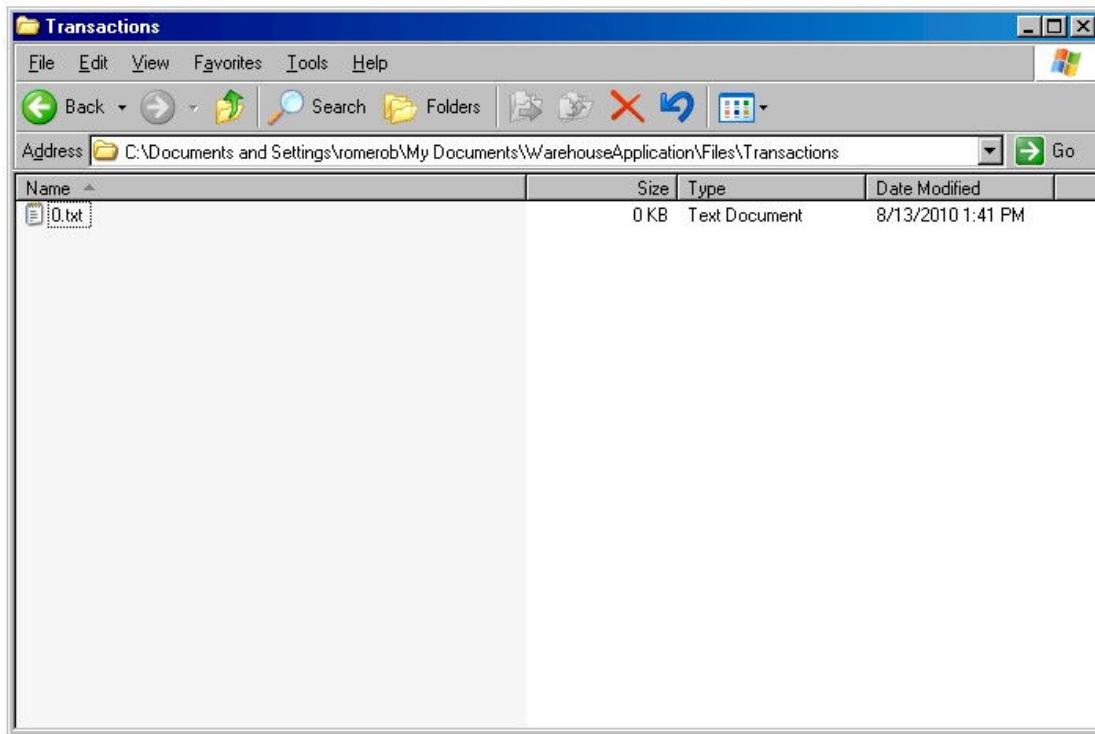
The Random Access File is initialised and the Indexes are written to the index files



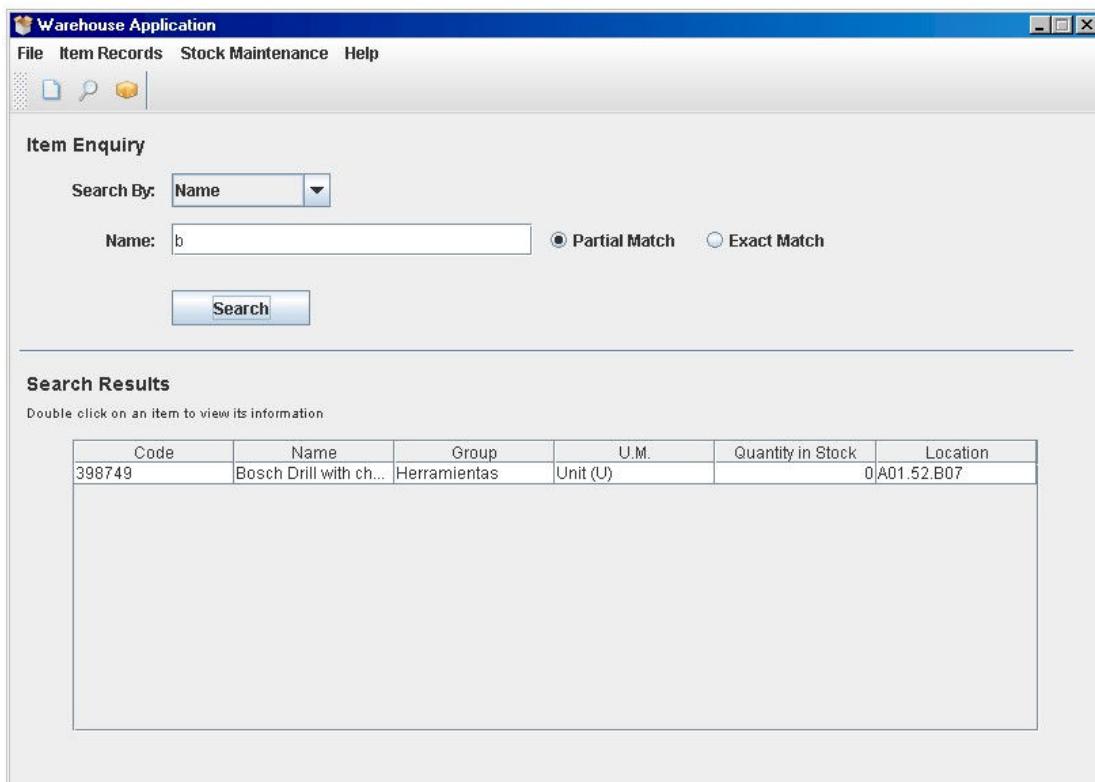
Name	Size	Type	Date Modified
Item_Images		File Folder	8/10/2010 11:32 AM
Transactions		File Folder	8/10/2010 11:32 AM
Groups.txt	1 KB	Text Document	7/2/2010 11:12 PM
Item_Code.txt	1 KB	Text Document	8/13/2010 1:37 PM
Item_Name.txt	1 KB	Text Document	8/13/2010 1:37 PM
Items.dat	5 KB	DAT File	8/13/2010 1:37 PM
UMs.txt	1 KB	Text Document	7/2/2010 11:00 PM
Users.txt	1 KB	Text Document	8/6/2010 5:31 PM



The empty transactions file for this item is created



The item shows up in the item list when searching for it by partial name with the query "b"

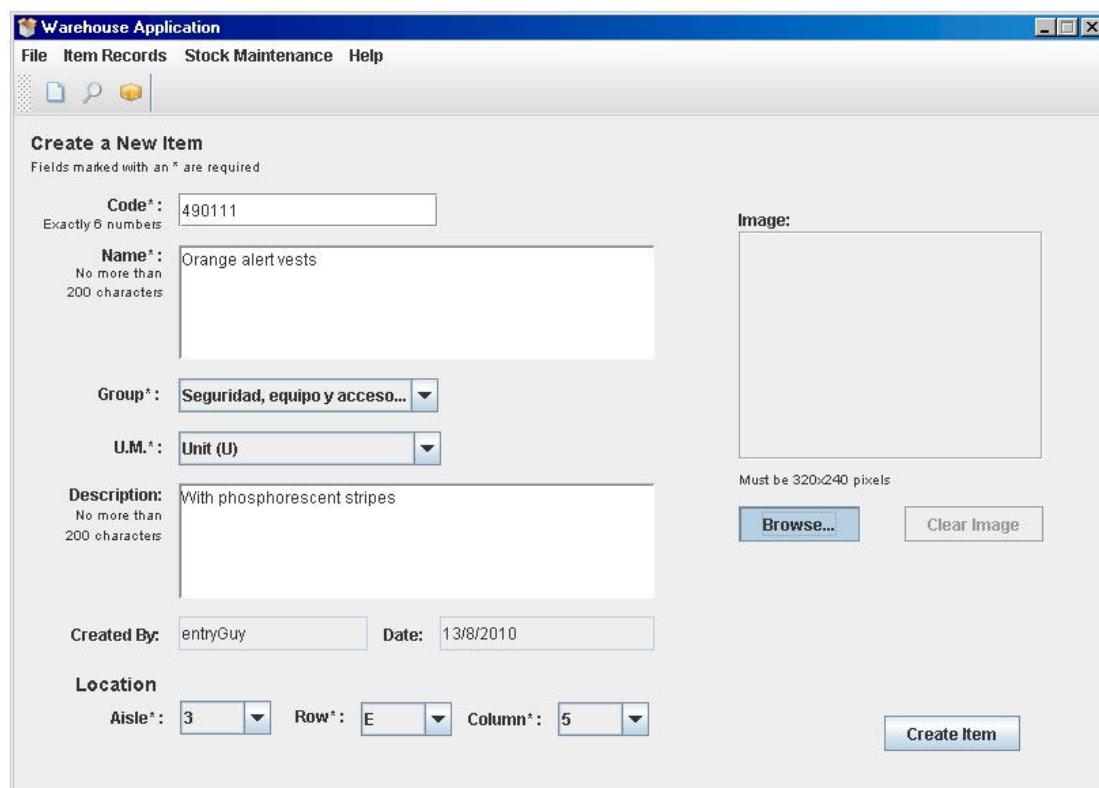


## 2. Creating an image and assigning a .jpg image

The above steps are repeated to create the item

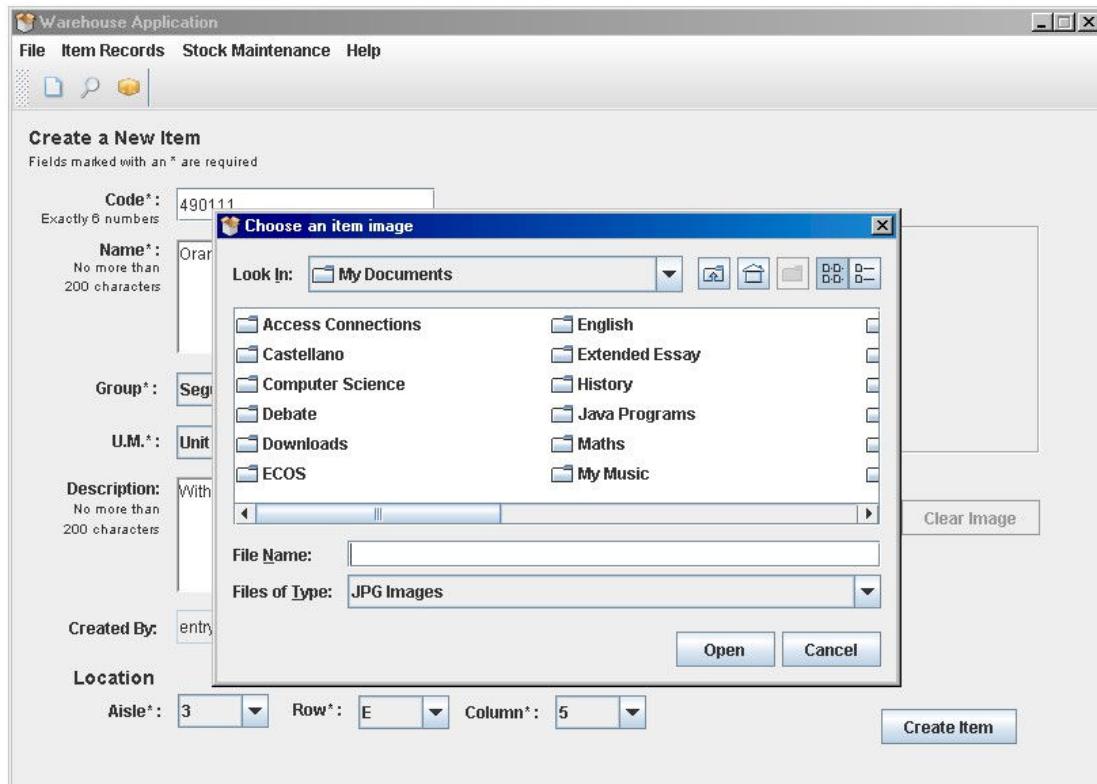
Code: 490111  
Name: Orange alert vests  
Group: Seguridad, equipo y accesorios  
U.M.: Unit (U)  
Description: With phosphorescent stripes  
Location:  
Aisle: 3  
Row: E  
Column: 5

Before the 'Create Item' button is pressed, the 'Browse' button is pressed to open the File Chooser to select an item image



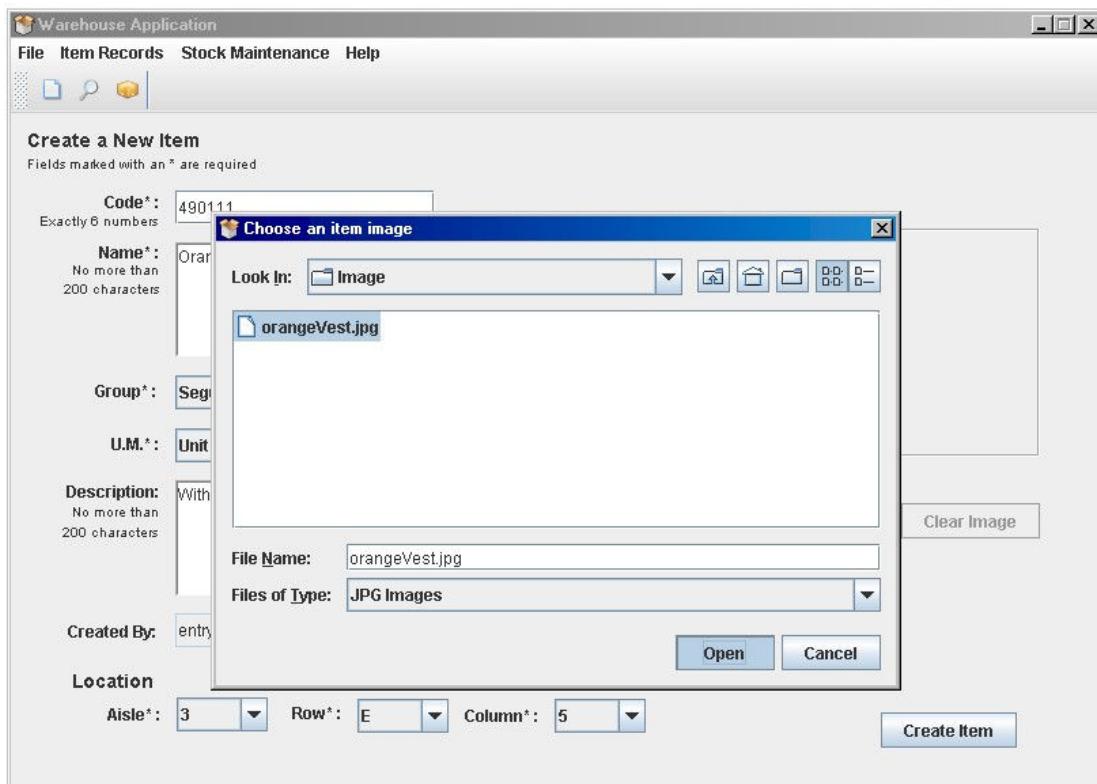
### Result

The File Chooser dialog opens



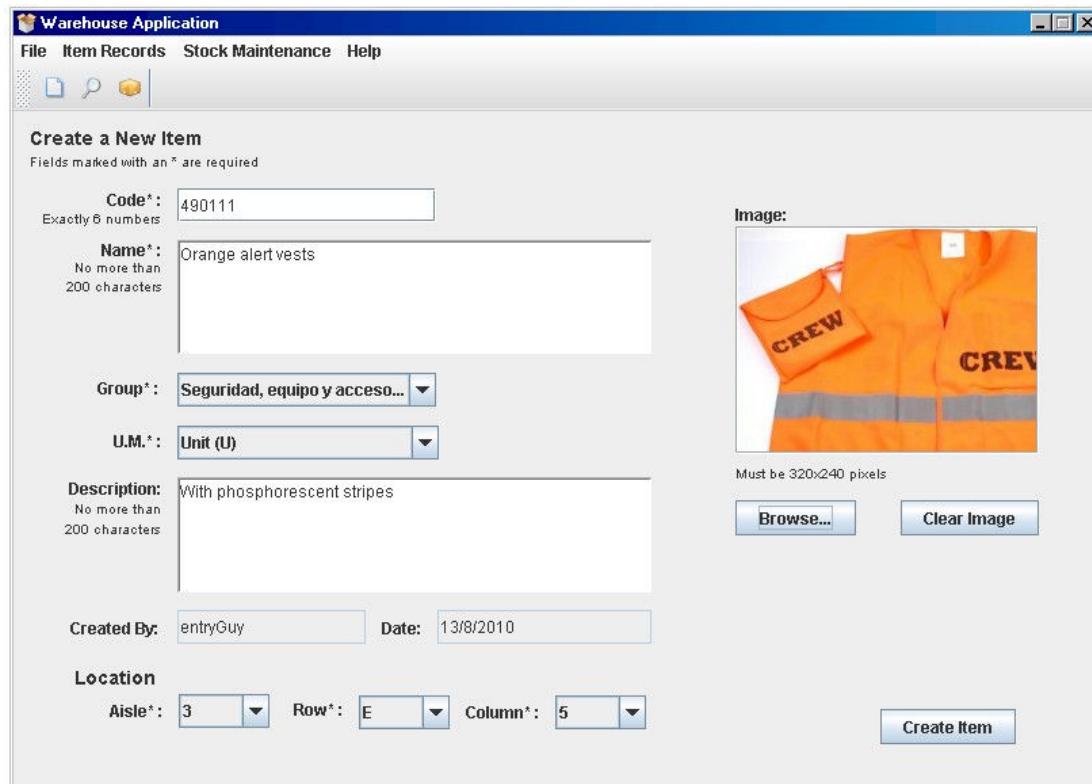
Note that only .jpg files (besides folders) can be selected

The image 'orangeVest.jpg' in the folder 'Image' is selected. The 'Open' button is pressed



### Result

The image displays in the label. Note that the box only shows part of the image that can have a maximum size of 320x240px

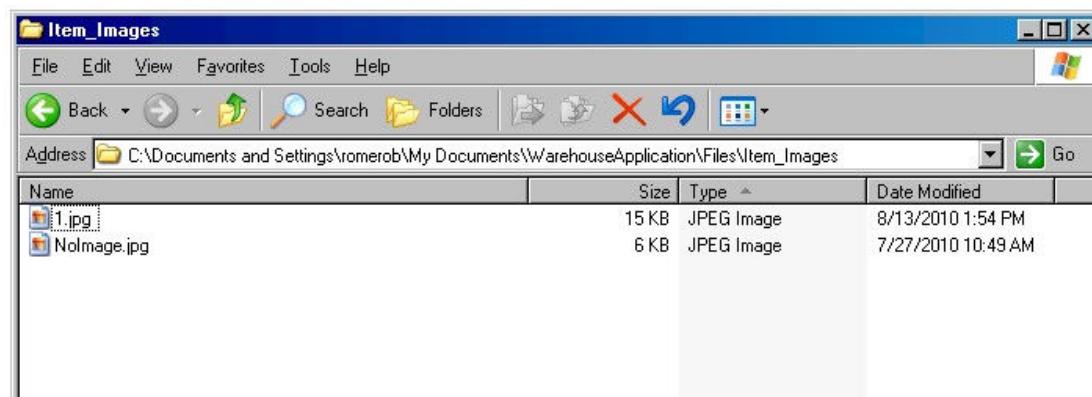


Note: only the proof directly related to adding an item image will be shown below.  
Refer above for the rest

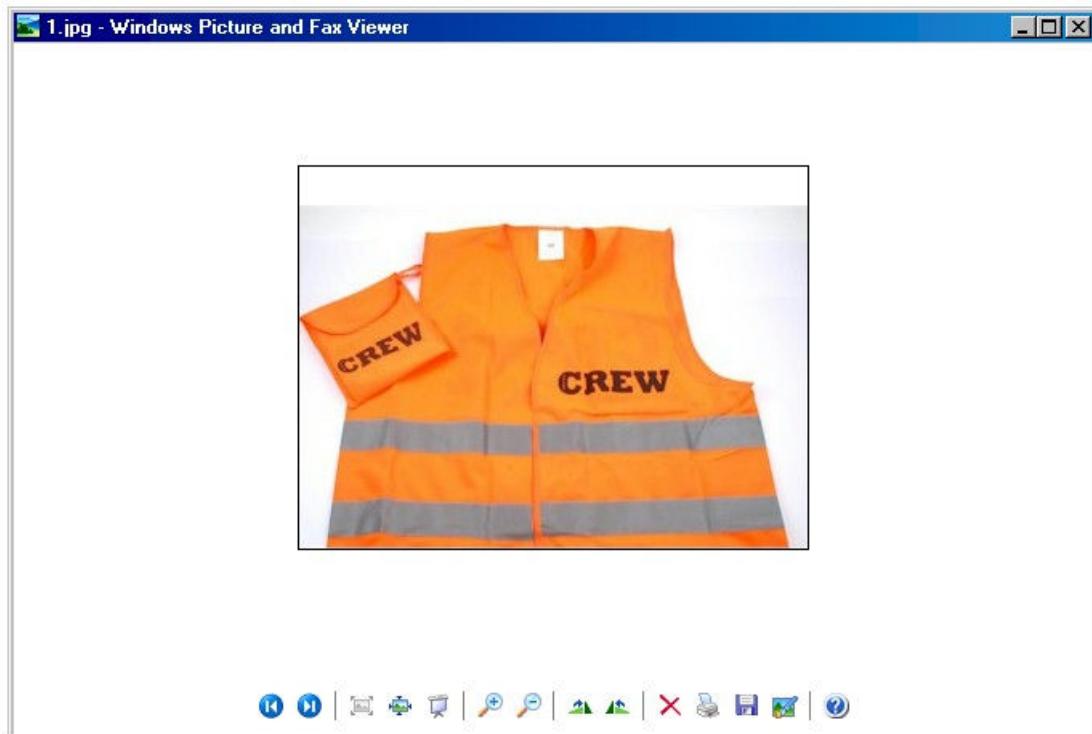
The 'Create Item' button is pressed

The user is informed that the process has been successful and the item is created in the files

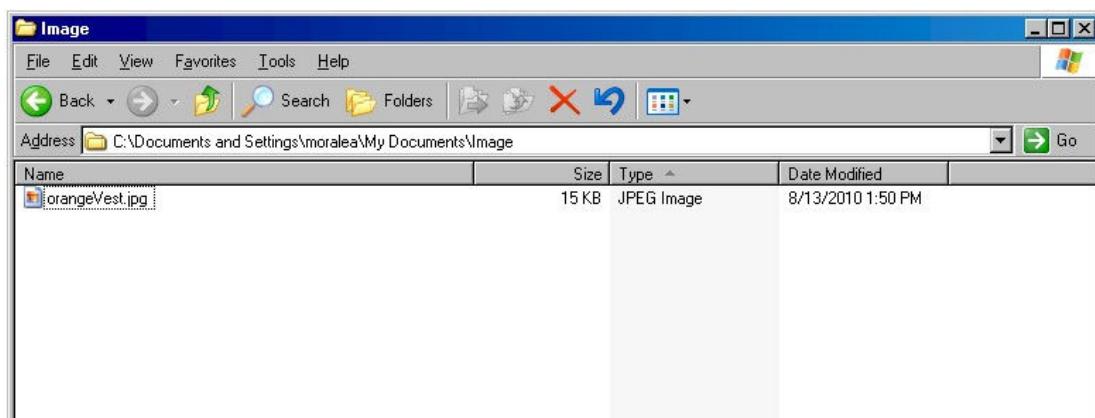
The item image is copied to the images folder, and renamed to the Item ID.



Opening the image, the correct image displays:



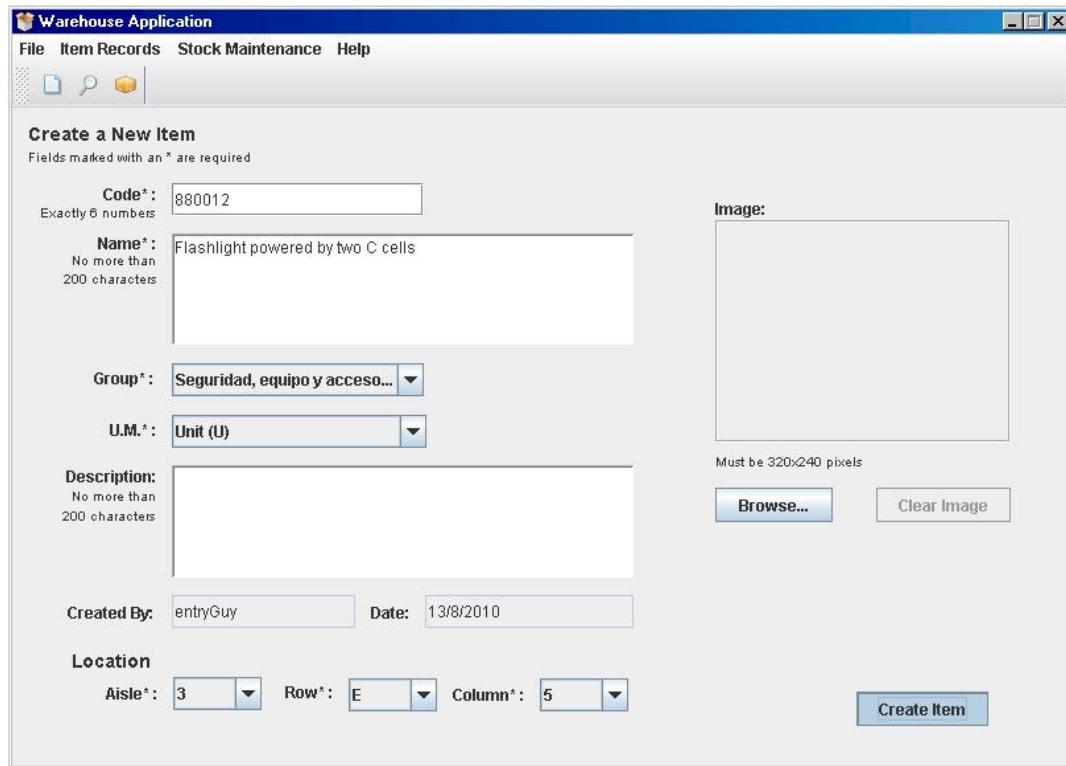
The item still exists in its previous location



### 3. Creating and adding an item into a taken location

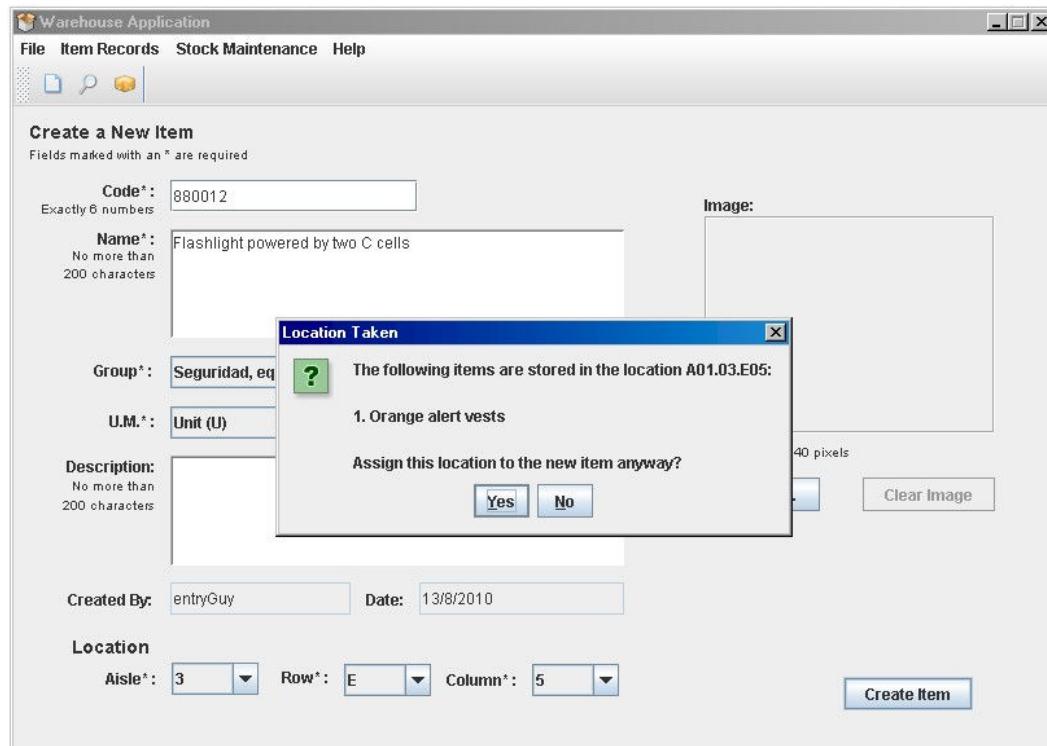
Code: 880012  
Name: Flashlight powered by two C cells  
Group: Seguridad, equipo y accesorios  
U.M.: Unit (U)  
Description:  
Location:  
    Aisle: 3  
    Row: E  
    Column: 5

The 'Create Item' button is pressed



### Result

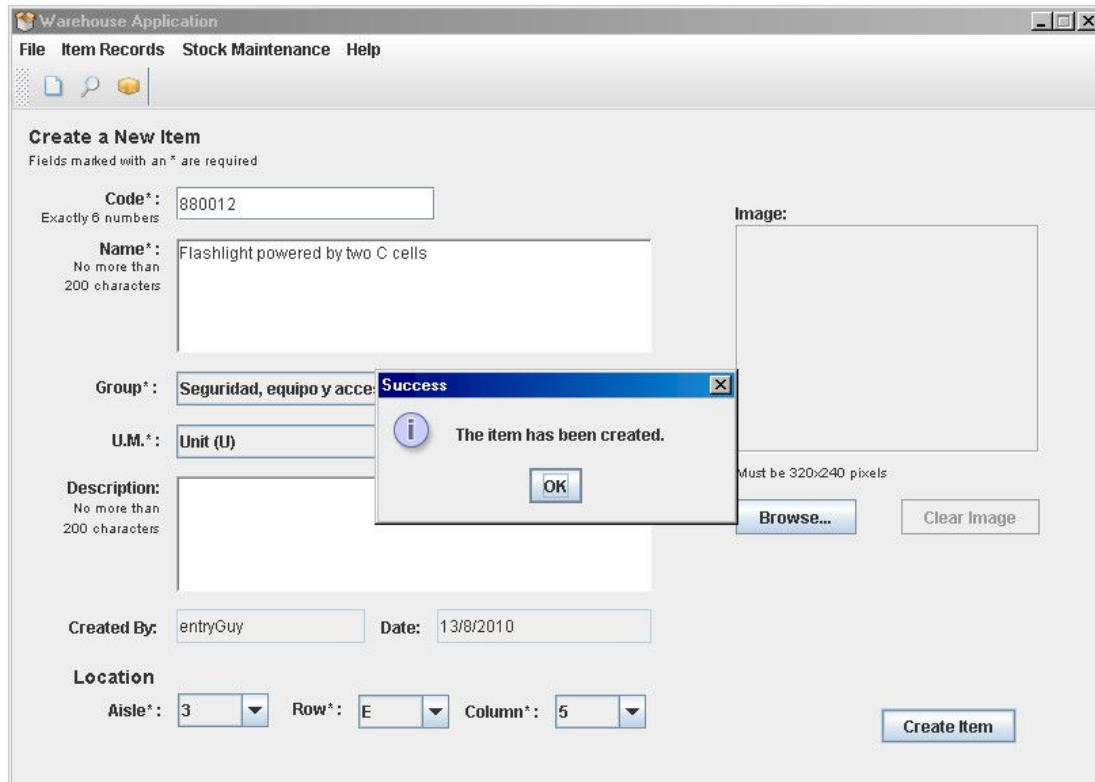
The user is informed that another item(s) are stored in the location, and asked if the creation process should proceed



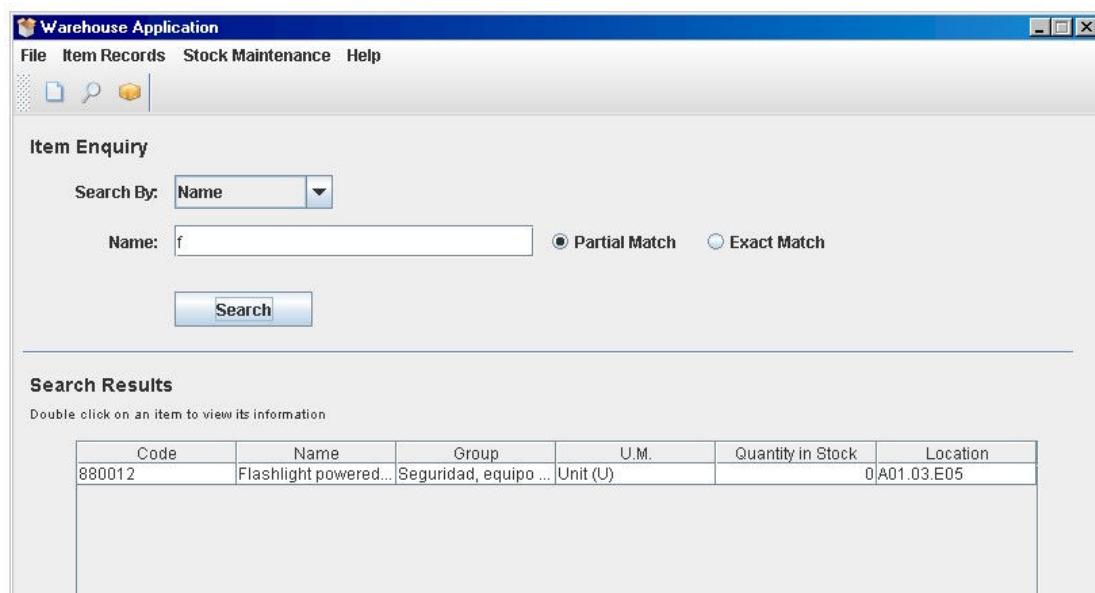
EITHER Press 'Yes'

*Result*

The item is created  
(Writing to files shown above)



The item shows up in a partial search by name with the query 'f'



OR Press 'No'

*Result*

The item is not created, and focus returns to the item creation screen, where the user might choose to change some information

The screenshot shows the 'Create a New Item' window of the Warehouse Application. The window title is 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', and 'Help'. The toolbar has icons for New, Open, Save, and Print. The main area is titled 'Create a New Item' and contains the following fields:

- Code\***: 880012 (labeled 'Exactly 6 numbers')
- Name\***: Flashlight powered by two C cells (labeled 'No more than 200 characters')
- Group\***: Seguridad, equipo y acceso...
- U.M.\***: Unit (U)
- Description**: (empty text area labeled 'No more than 200 characters')
- Created By**: entryGuy
- Date**: 13/8/2010
- Location**: Aisle\*: 3, Row\*: E, Column\*: 5
- Image**: (Empty image placeholder with instructions: 'Must be 320x240 pixels', 'Browse...', 'Clear Image')
- Create Item** button

**INCORRECT INPUT**

1. A code longer than 6 digits

Code: 9084383434 [10 digits]

The above is entered into the Code text field

*Result*

The code is truncated to 6 digits

The screenshot shows the 'Create a New Item' window of the Warehouse Application. The window title is 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', and 'Help'. The toolbar has icons for New, Open, Save, and Print. The main area is titled 'Create a New Item' and contains the following fields:

- Code\***: 908438 (labeled 'Exactly 6 numbers')
- Name\***: (empty text area labeled 'No more than 200 characters')
- Image**: (Empty image placeholder)

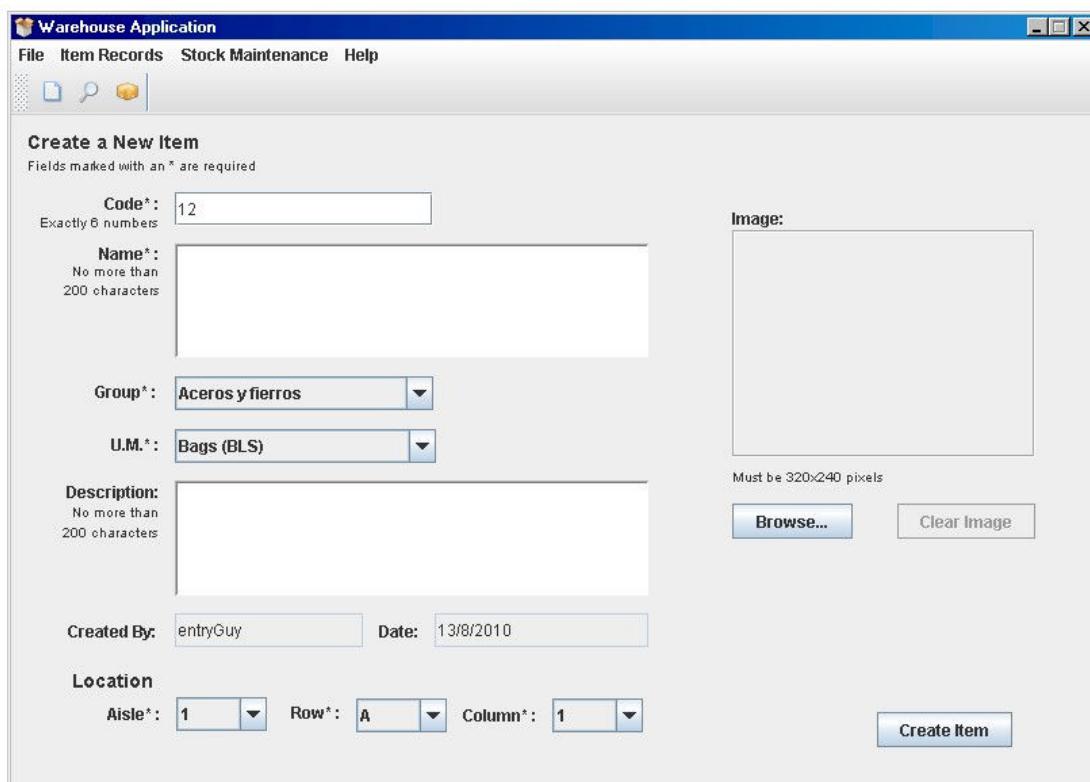
2. A code longer containing non-digits

Code: 12abcd

The above is entered into the Code text field

*Result*

Only digits can be input



3. An item name or description longer than 200 characters

Name:

aa  
aa  
aa  
aa [210 characters]

Description:

aa  
aa  
aa  
aaa [210 characters]

The above are entered into the name and description text fields respectively

*Result*

The name and description truncate to 200 characters

The screenshot shows the 'Warehouse Application' interface with the title 'Create a New Item'. The 'Name\*' field contains a long string of 'aaaa' characters, truncated to 200 characters as specified in the validation message 'No more than 200 characters'. The 'Description:' field also contains a similar truncated string. Other fields like 'Code\*', 'Group\*', 'U.M.\*', 'Created By.', and 'Location' are populated with valid data. An 'Image:' section is present but empty.

4. Not all required fields have been completed

Code: [blank]  
Name: [blank]  
Group: Electronico  
U.M.: Unit (U)  
Description:  
Location:  
    Aisle: 24  
    Row: A  
    Column: 8

The 'Create Item' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Help

Create a New Item  
Fields marked with an \* are required

Code\*: Exactly 6 numbers

Name\*: No more than 200 characters

Group\*: Electronico

U.M.\*: Unit (U)

Description: No more than 200 characters

Image: Must be 320x240 pixels

Browse... Clear Image

Created By: entryGuy Date: 13/8/2010

Location  
Aisle\*: 24 Row\*: A Column\*: 8

### Result

The user is informed that some fields are missing

**Warehouse Application**

File Item Records Stock Maintenance Help

Create a New Item  
Fields marked with an \* are required

Code\*: Exactly 6 numbers

Name\*: No more than 200 characters

Group\*: Electronico

U.M.\*: Unit (U)

Description: No more than 200 characters

Image: Must be 320x240 pixels

Browse... Clear Image

Some fields are missing

OK

Created By: entryGuy Date: 13/8/2010

Location  
Aisle\*: 24 Row\*: A Column\*: 8

## Growing the Items Random Access File

As specified in the code, when first initialised the Items random access file has a maximum capacity of 10 records. When there is only one empty record, the record capacity is grown by a factor of 1.5. This functionality will be demonstrated.

9 random items are created. They are assigned the same group, 'Aceros y Fierros', to show that the file contains 9 items.

*The 9 random items:*

The screenshot shows the 'Warehouse Application' window. In the 'Item Enquiry' section, 'Search By' is set to 'Group' and 'Group' is set to 'Aceros y fierros'. A 'Search' button is present. Below this, the 'Search Results' section displays a table with 9 rows of item information:

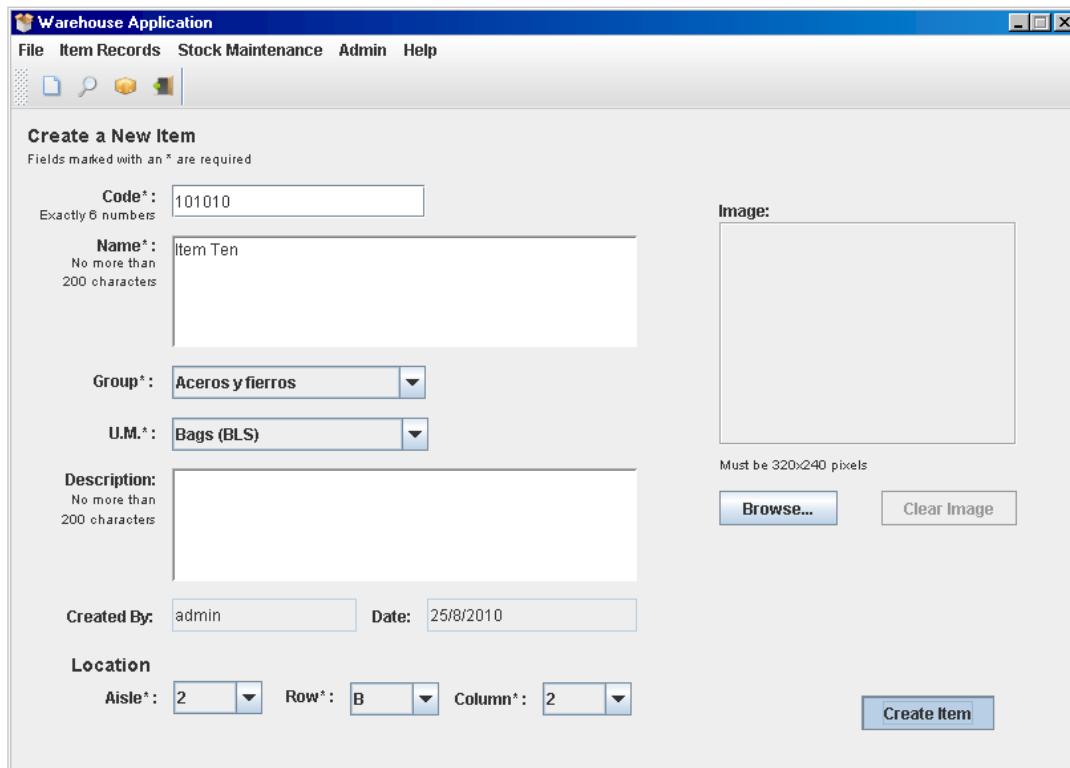
Code	Name	Group	U.M.	Quantity in Stock	Location
111111	Item One	Aceros y fierros	Bags (BLS)		0A01.01.A01
222222	Item Two	Aceros y fierros	Bags (BLS)		0A01.01.A01
333333	Item Three	Aceros y fierros	Bags (BLS)		0A01.01.A01
444444	Item Four	Aceros y fierros	Bags (BLS)		0A01.01.A01
555555	Item Five	Aceros y fierros	Bags (BLS)		0A01.01.A01
666666	Item Six	Aceros y fierros	Bags (BLS)		0A01.01.A01
777777	Item Seven	Aceros y fierros	Bags (BLS)		0A01.01.A01
888888	Item Eight	Aceros y fierros	Bags (BLS)		0A01.01.A01
999999	Item Nine	Aceros y fierros	Bags (BLS)		0A01.01.A01

The size of the 'Items.dat' file before the file is grown is 9 KB

The screenshot shows a Windows Explorer window titled 'Files'. The address bar shows the path 'C:\Documents and Settings\romerob\My Documents\WarehouseApplication\Files'. The contents of the folder are listed in a table:

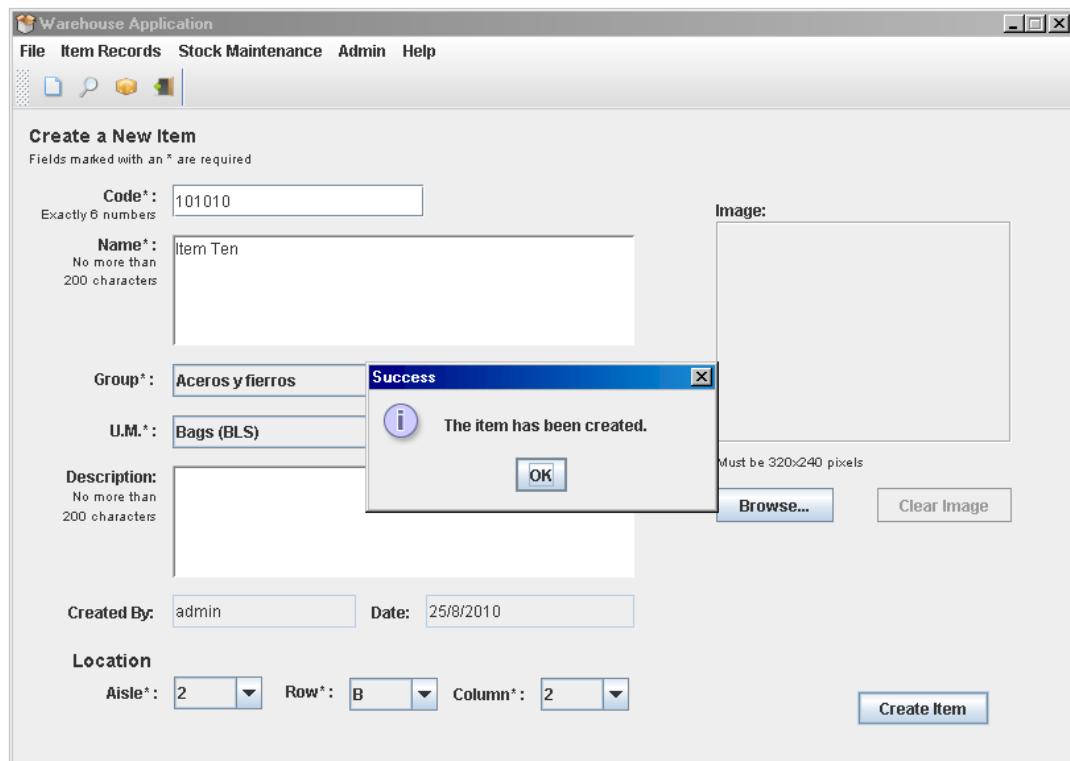
Name	Size	Type	Date Modified
Item_Images		File Folder	8/25/2010 6:30 PM
Transactions		File Folder	8/25/2010 6:37 PM
Groups.txt	1 KB	Text Document	7/2/2010 11:12 PM
Item_Code.txt	1 KB	Text Document	8/25/2010 6:37 PM
Item_Name.txt	1 KB	Text Document	8/25/2010 6:37 PM
Items.dat	9 KB	DAT File	8/25/2010 6:37 PM
UMs.txt	1 KB	Text Document	7/2/2010 11:00 PM
Users.txt	1 KB	Text Document	8/24/2010 10:01 PM

A 10<sup>th</sup> item is created, and the 'Create Item' button is pressed

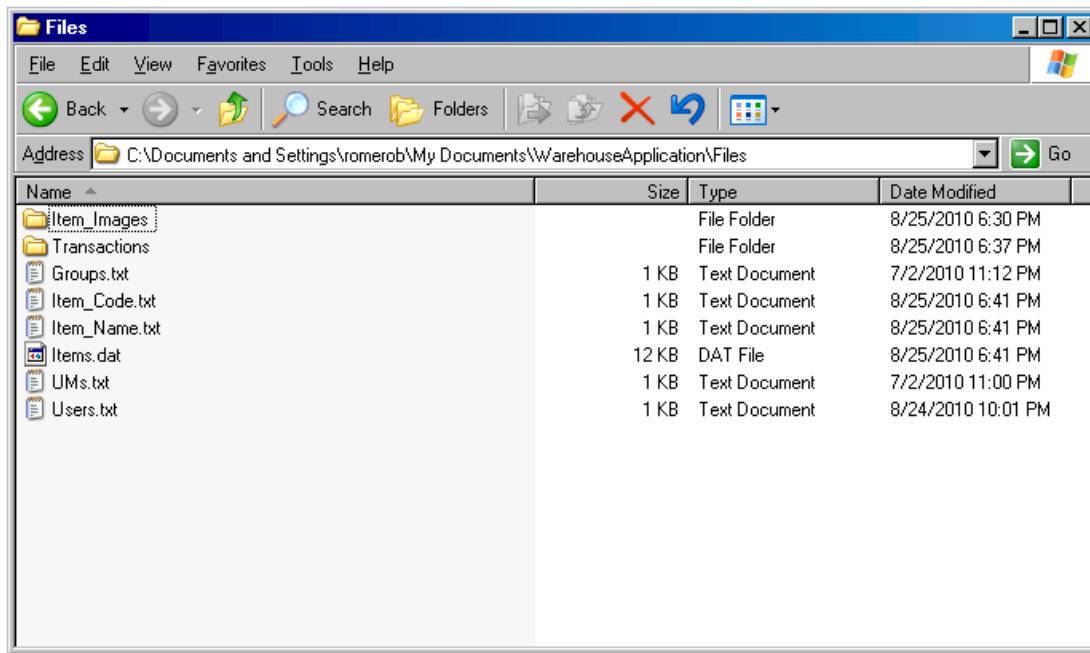


### Result

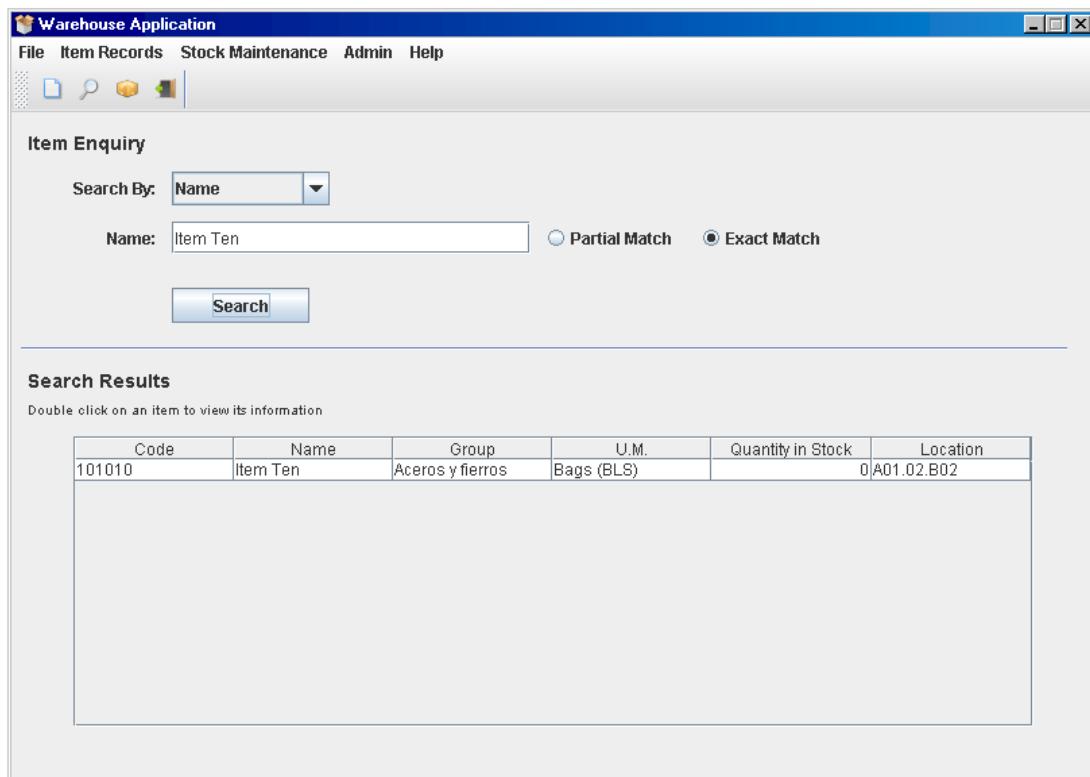
The user is informed that the item was created



The 'Items.dat' file is grown, its size is now 12 KB



The created item, of name 'Item Ten', shows up when a search for it is done



### Accessing an item in the middle of the file

To demonstrate that random access to the items file is properly functioning, an item in the middle of the file will be accessed by searching for it and loading its description.

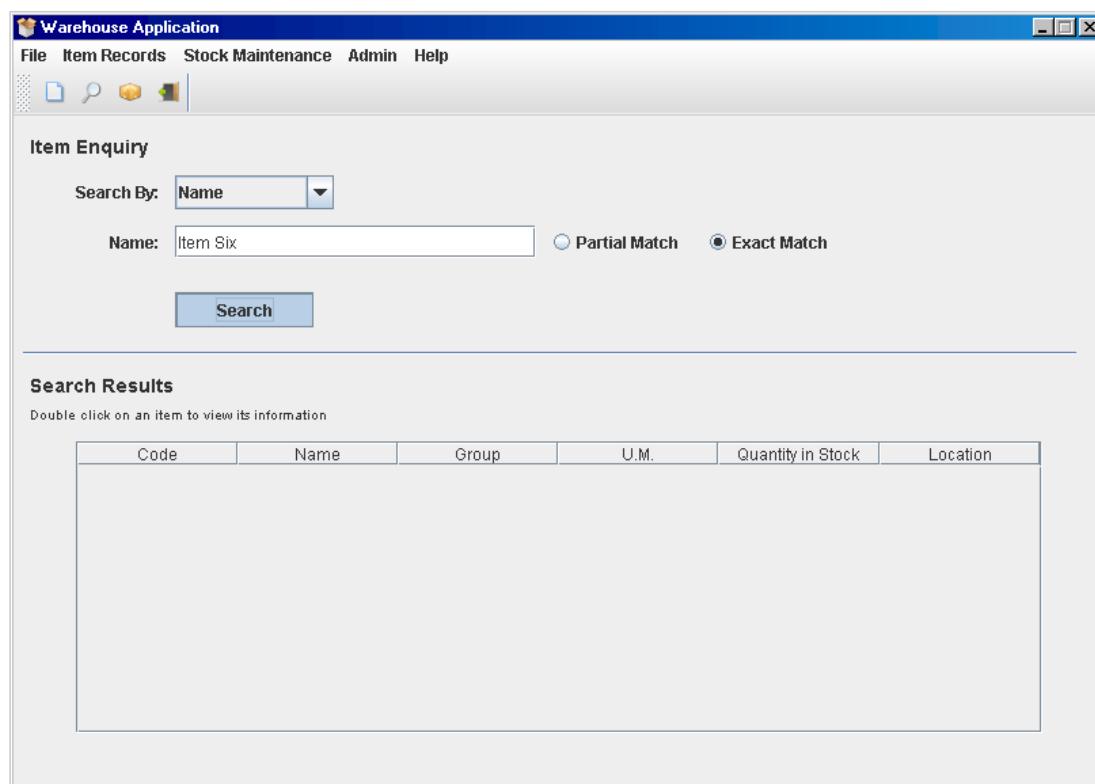
The same items shown in the example above remain. The item of name 'Item Six' is in position 5 in the random access file as shown by the index by name file.



```
Item One|0
Item Two|1
Item Three|2
Item Four|3
Item Five|4
Item Six|5
Item Seven|6
Item Eight|7
Item Nine|8
Item Ten|9
```

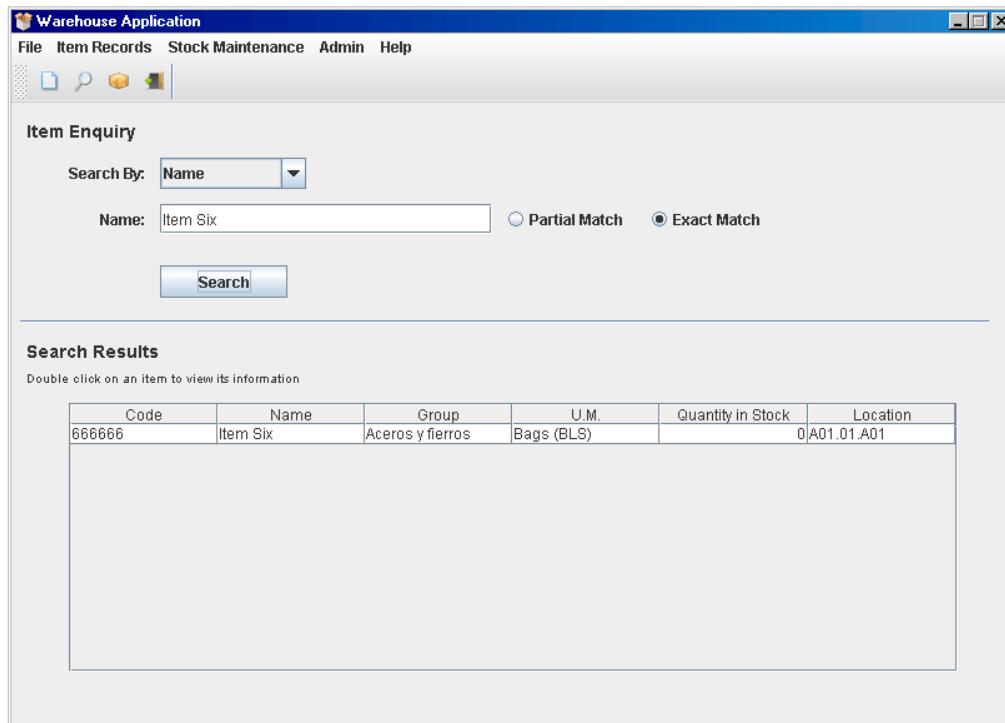
An exact search by name with the query 'Item Six' is carried out to see if the item can be accessed.

The 'Search' button is pressed



*Result*

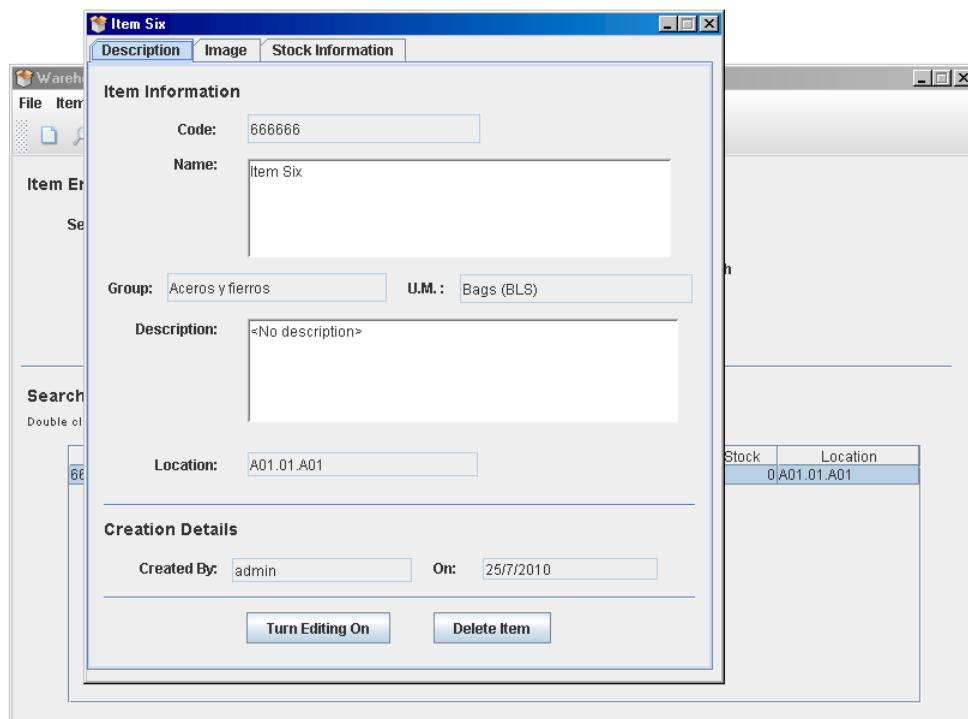
The item is displayed in the search results.



The item is double clicked.

*Result*

The correct Item Description screen is shown.

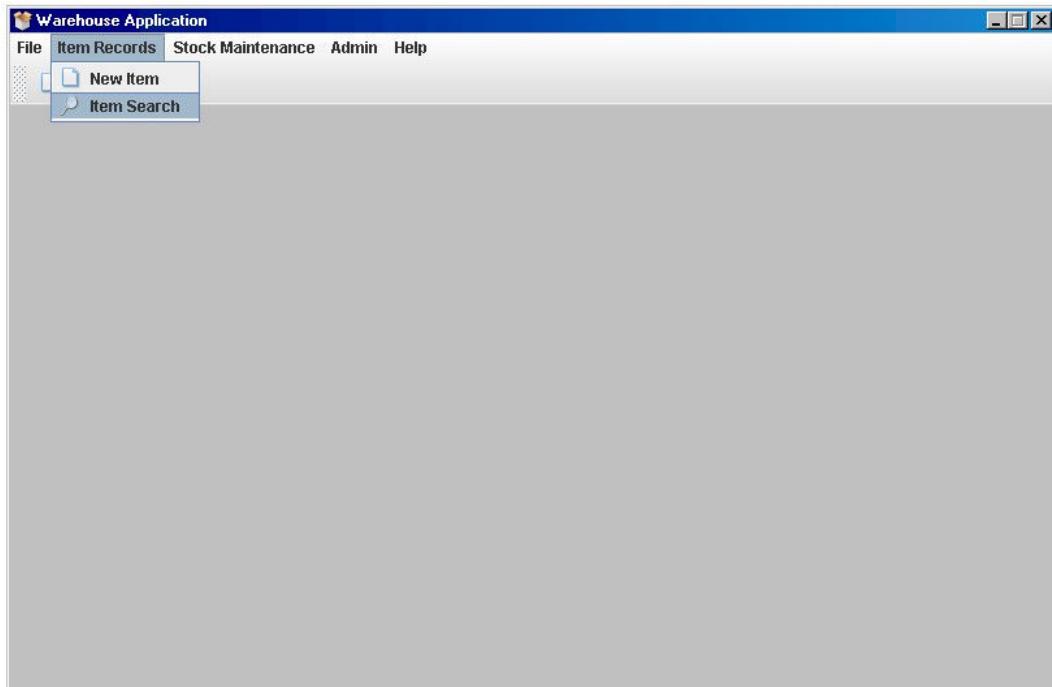


To demonstrate the application's search capabilities, the following items have been created

<b>Code</b>	<b>Name</b>	<b>Group</b>	<b>Location</b>
1. 013294	Laptop IBM ThinkPad R52 for Administrative Use	Electronico	A01.06.B.06
2. 220018	Lantern for Mining Excavations	Herramientas	A01.52.D.10
3. 709125	Steel Nails in 1/16" width	Aceros y Fierros	A01.28.A.05
4. 559207	1/3" Iron Bolts	Aceros y Fierros	A01.23.E.08
5. 409621	Synthetic Glue for Wood Sheets	Varios	A01.40.C.06
6. 301298	1/2" wide nylon rope	Varios	A01.40.C.06
7. 660192	Left Handed Wire Cutters	Herramientas	A01.12.B.01

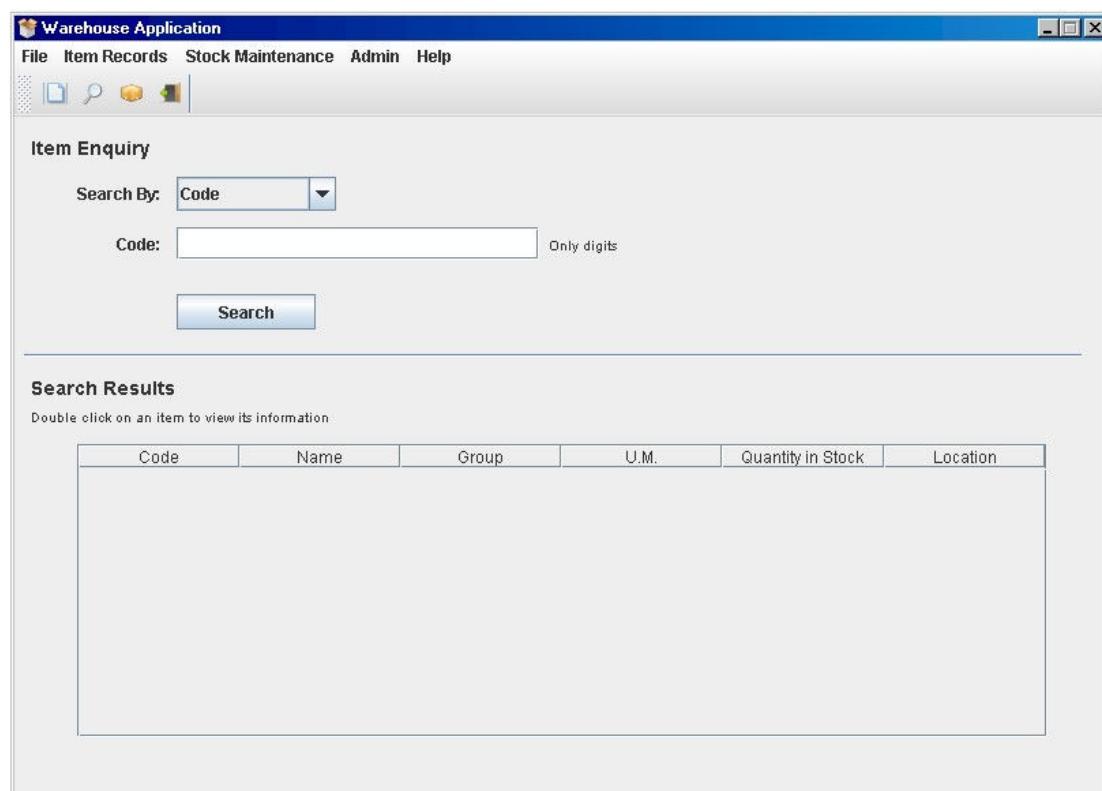
## LOADING THE ITEM SEARCH SCREEN

Clicking Item Records > Item Search from the Menu Bar



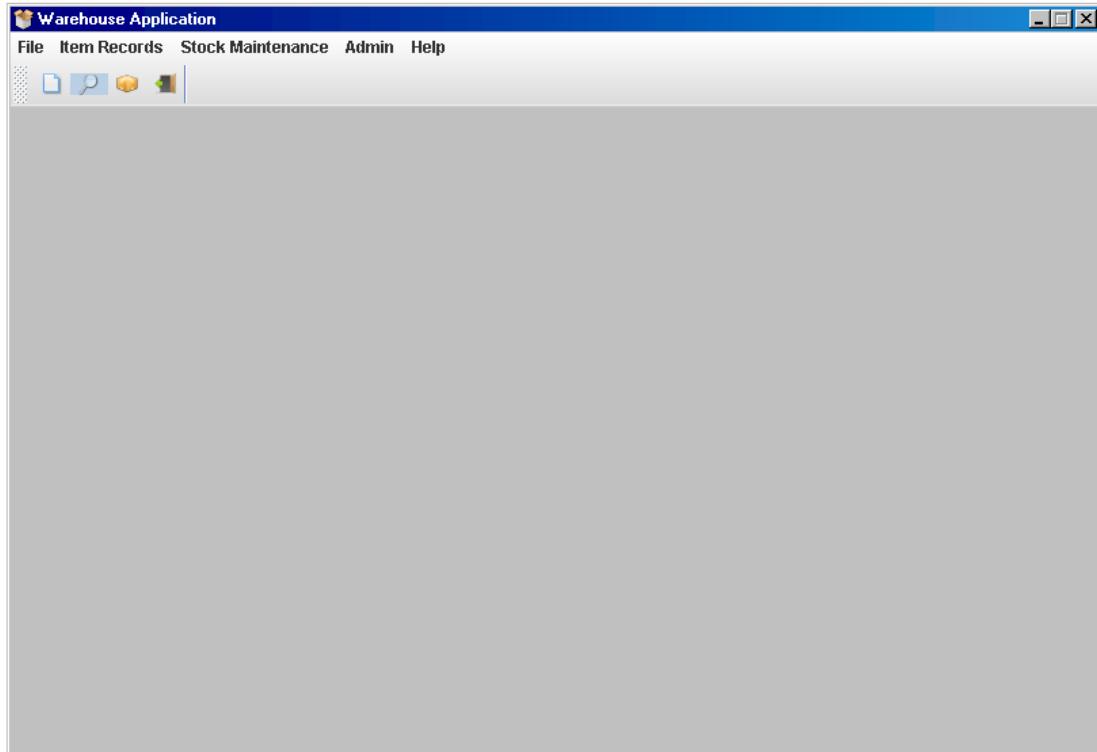
### *Result*

Item Search screen loads



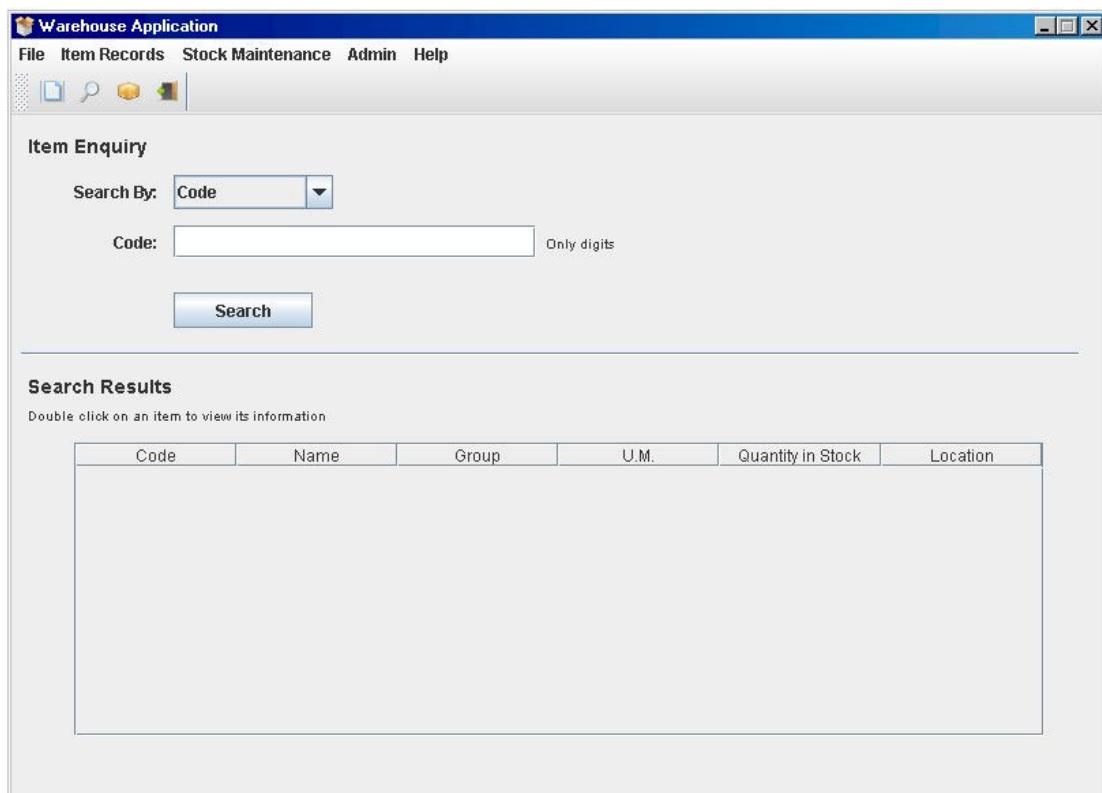
OR

Clicking the Item Search button in the Toolbar



*Result*

Item Search screen loads



## SEARCHING FOR AN ITEM BY CODE

Select Code from the drop-down list and the code search boxes load

### CORRECT INPUT

1. Searching for an item that exists

*Test*

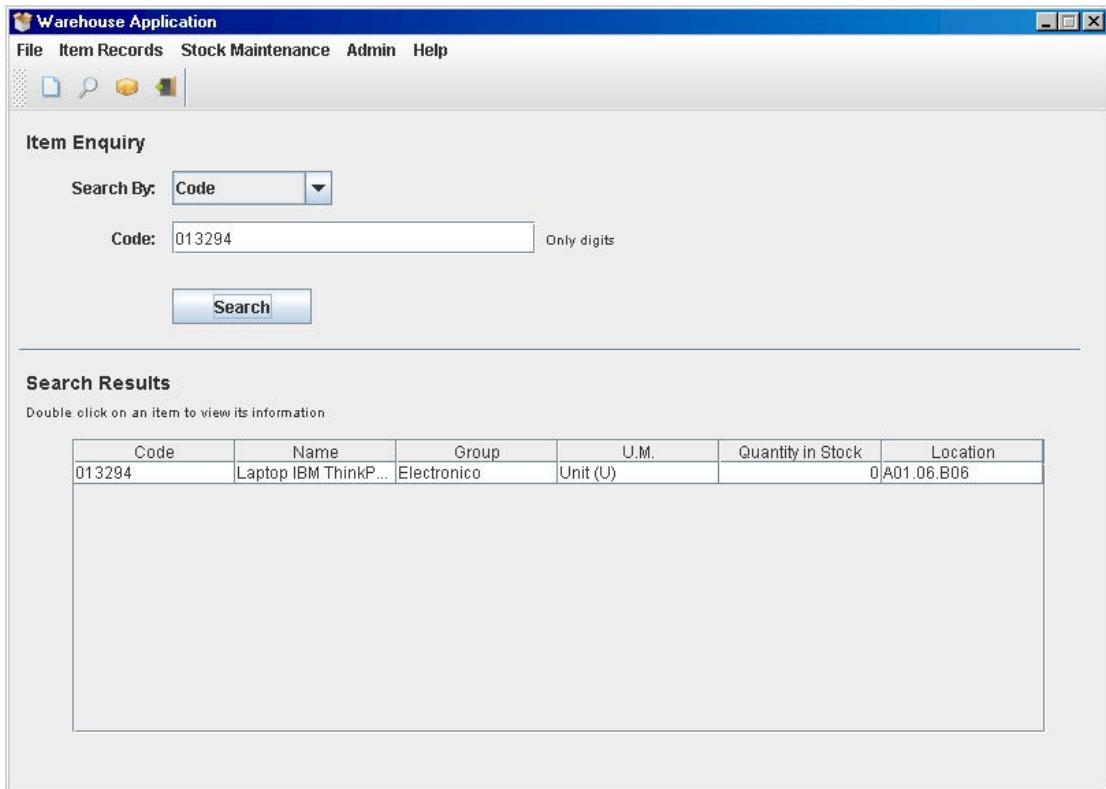
Search by:      Code  
Code:            013294                                 [the search query]

The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", "Admin", and "Help". Below the menu is a toolbar with icons for file operations. The main area is titled "Item Enquiry". It has a dropdown menu "Search By:" set to "Code" and a text input field "Code:" containing "013294". A note "Only digits" is displayed next to the input field. A "Search" button is below the input field. Below this is a section titled "Search Results" with the instruction "Double click on an item to view its information". A table header row is visible with columns: Code, Name, Group, U.M., Quantity in Stock, and Location.

The 'Search' button is clicked

*Results*

Appropriate result is displayed.

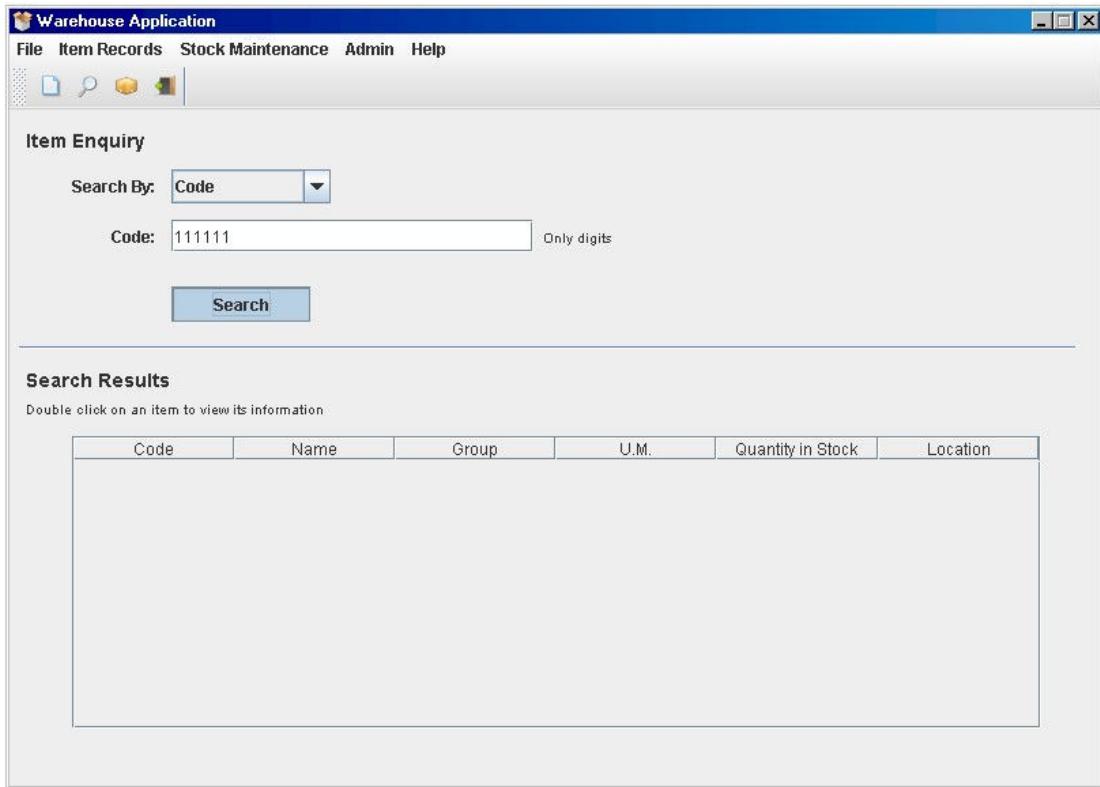


## 2. Searching for an item that does not exist

*Test*

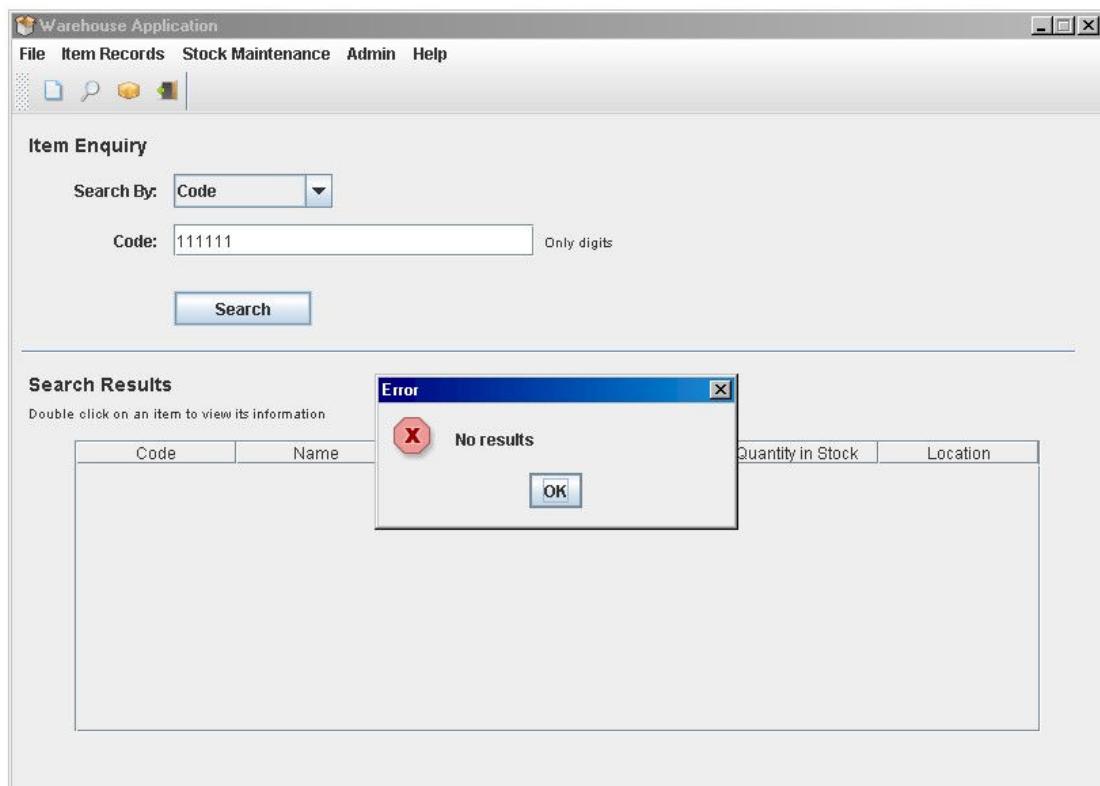
Search by: Code  
Code: 111111 [the search query]

The 'Search' button is pressed



### Result

User is informed that no items have been found



## INCORRECT INPUT

1. A query containing non-digits

Test:

Search by: Code  
Name: ab1234

*Result*

Non-digit characters are not entered

The screenshot shows the 'Warehouse Application' window with the title bar 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for file operations. The main area is titled 'Item Enquiry' and contains a form with the following fields:

- 'Search By:' dropdown set to 'Code'
- 'Code:' text input field containing '1234' with the validation message 'Only digits' displayed next to it.
- 'Search' button

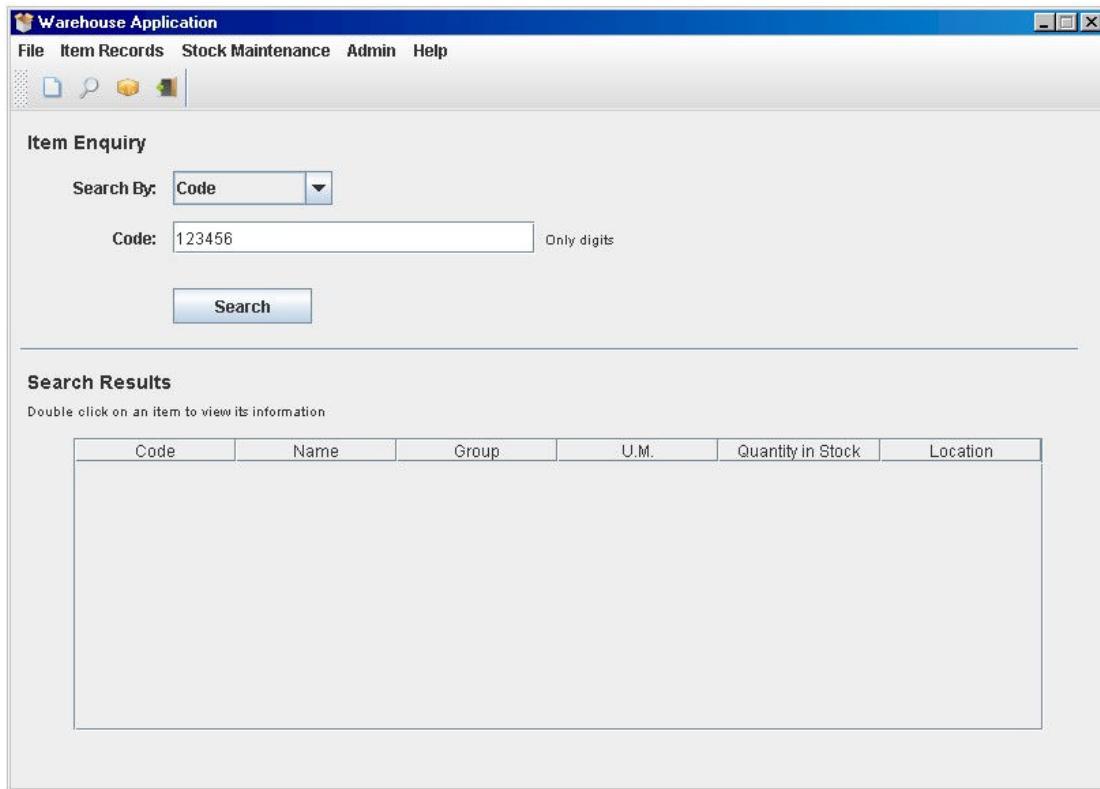
Below the form is a section titled 'Search Results' with the instruction 'Double click on an item to view its information'. A table header is visible but no data rows are present.

2. A query longer than 6 digits

Test:

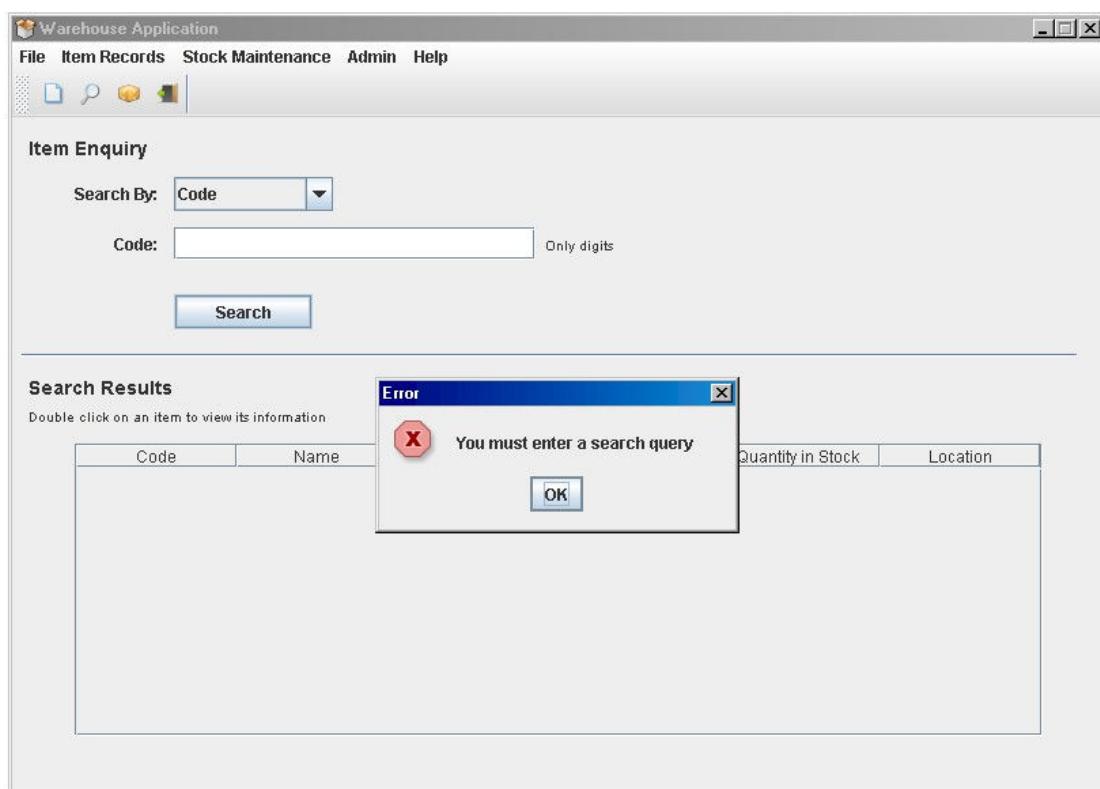
Search by: Code  
Name: 12345678 [8 digits]

*Result*



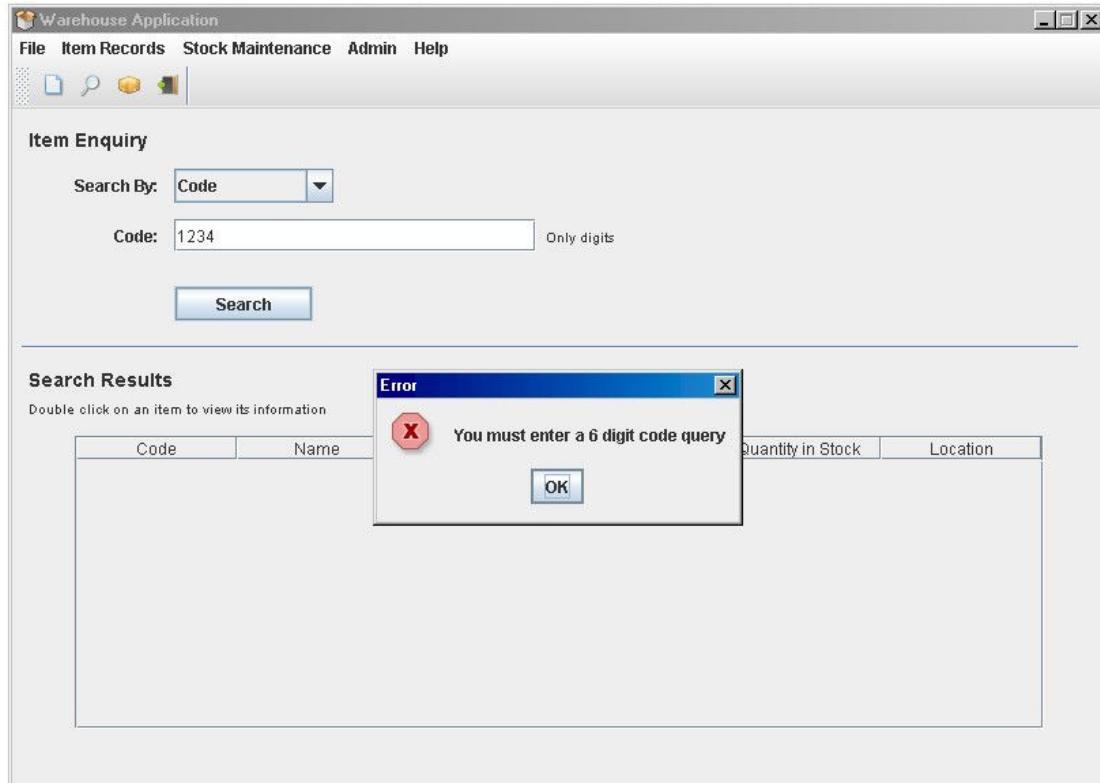
### 3. An empty query

*Result*



4. A query under 6 characters

Search by: Code  
Name: 1234 [4 digits]



## SEARCHING FOR AN ITEM BY PARTIAL NAME

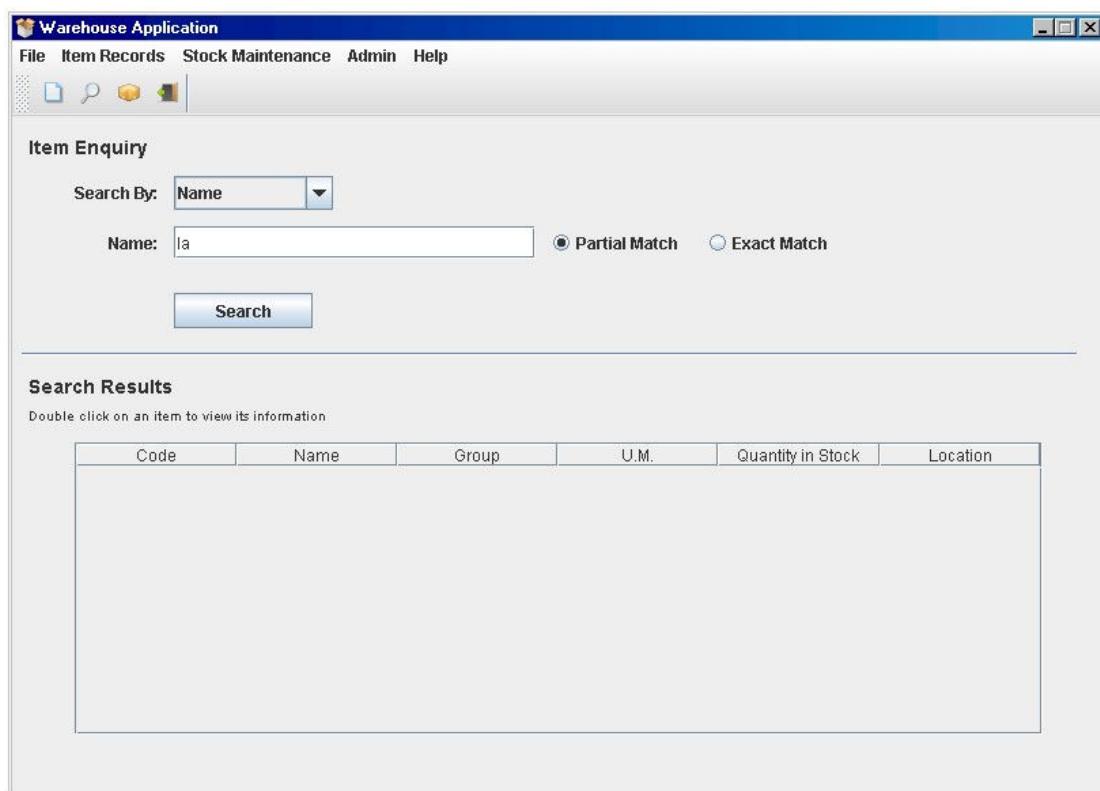
Select Name from the drop-down list and the name search boxes load

### CORRECT

Searching for an item that exists

*Test*

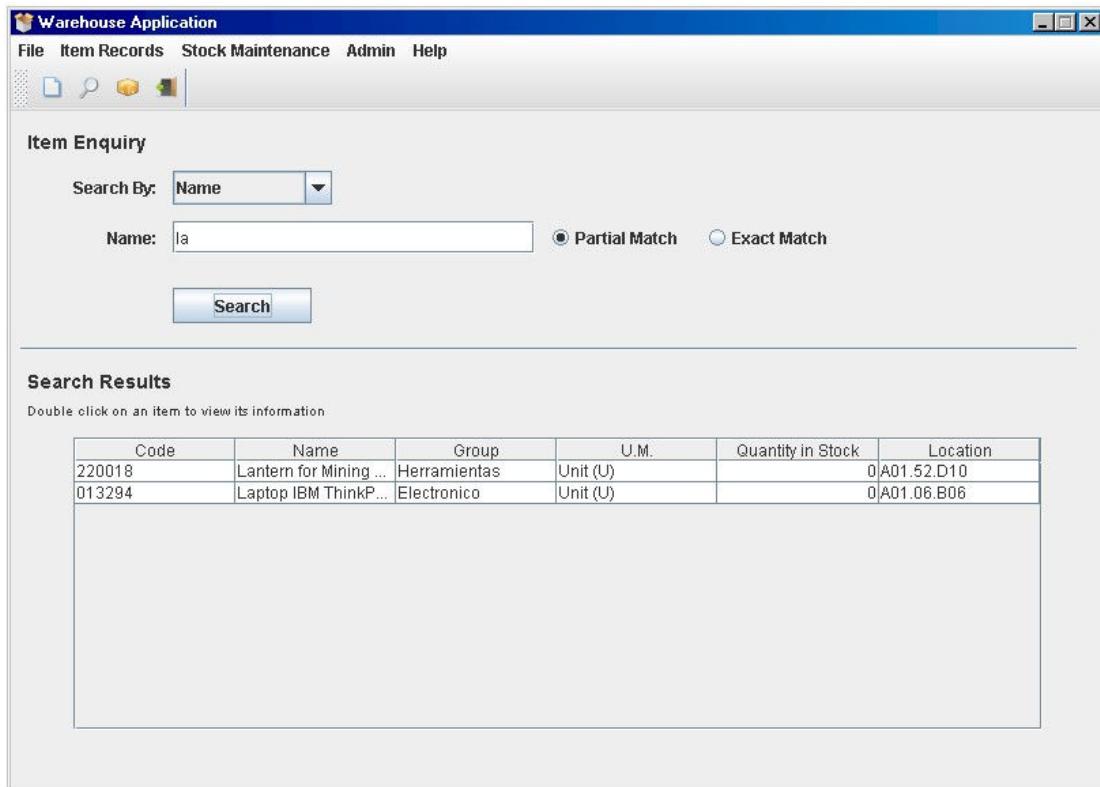
Search by: Name  
Name: la [the search query]  
Type: Partial Match



Clicking the 'Search' button

*Results*

Appropriate results display. Note that the search is NOT case sensitive

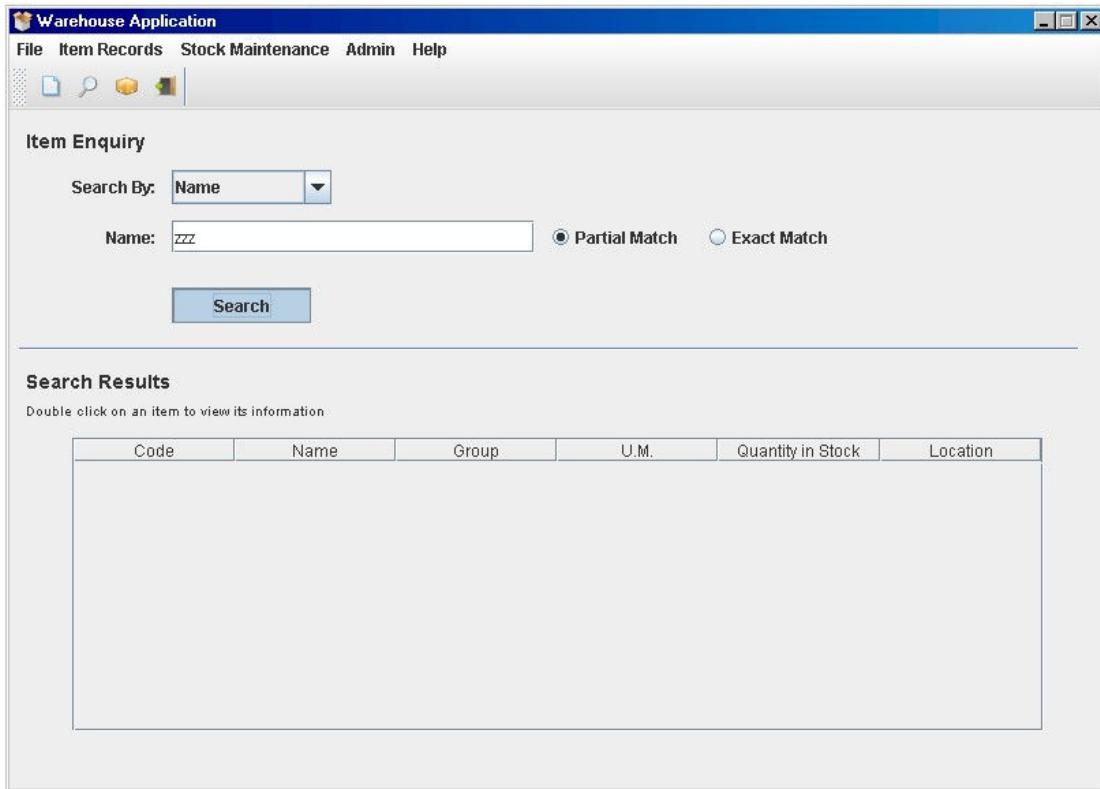


## 2. Searching for an item that does not exist

*Test*

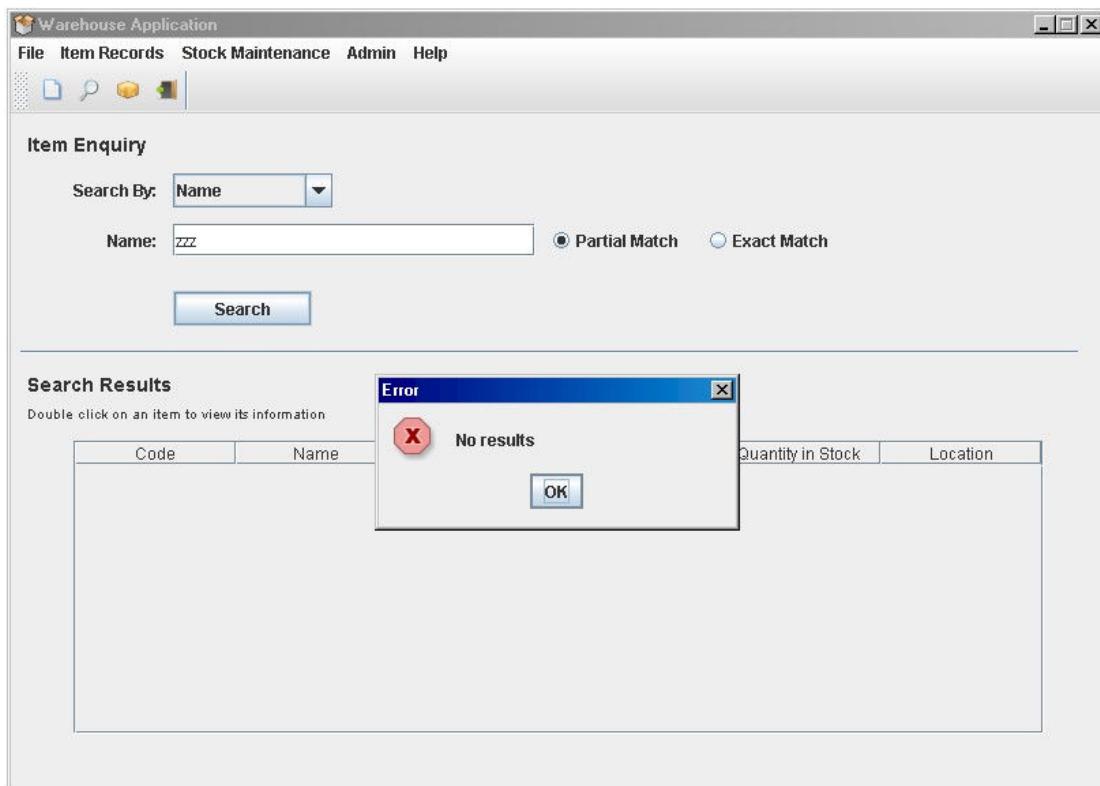
Search by: Name  
Name: zzz [the search query]  
Type: Partial Match

The search button is pressed



### Result

The user is informed that no items have been found



## **INCORRECT INPUT**

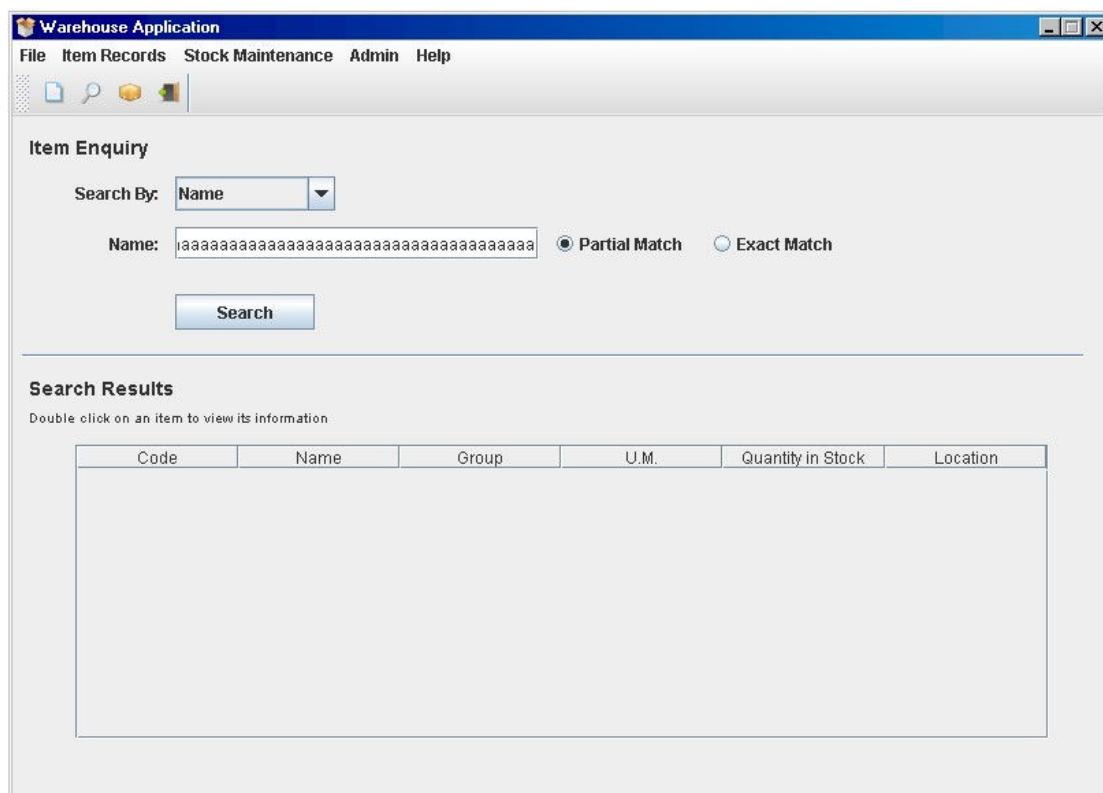
1. A query more than 200 characters in length

Test:

Search by: Name  
Name:  
aa  
aa  
aa  
aaa [210 characters]  
Type: Partial Match

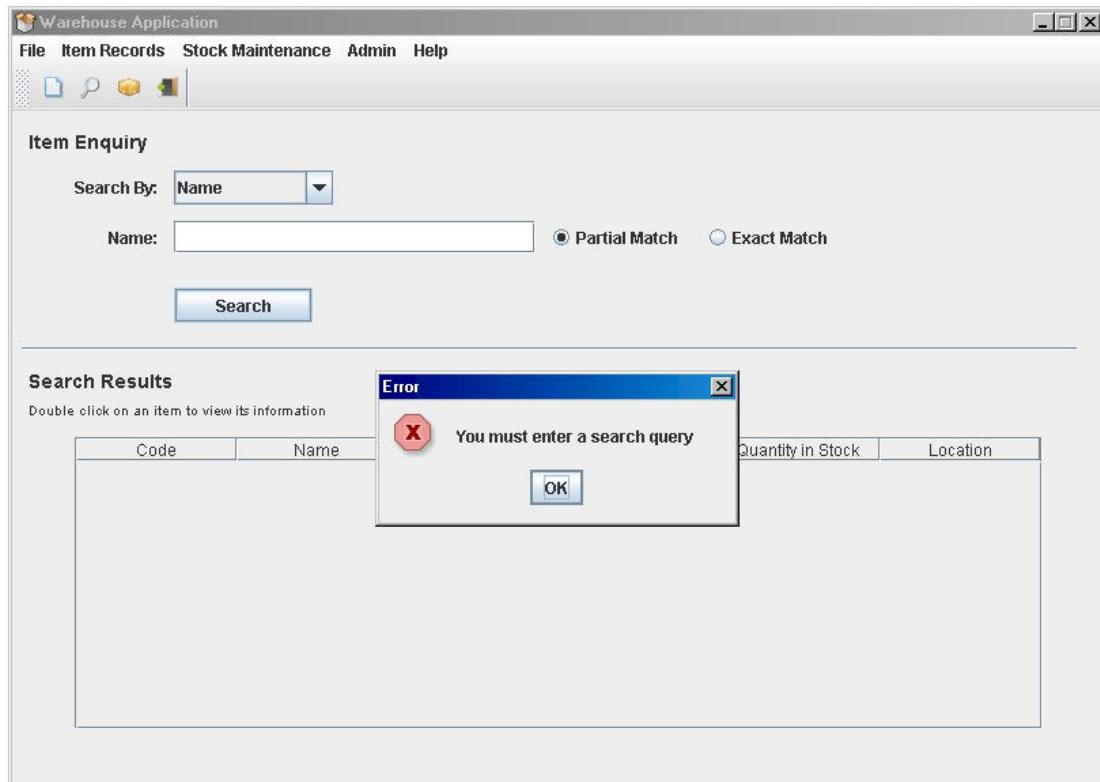
*Result*

The query truncates at 200 characters



2. No (i.e. empty) search query

*Result*



## SEARCHING FOR AN ITEM BY EXACT NAME

Select Name from the drop-down list and the name search boxes load

### CORRECT INPUT

Searching for an item that exists

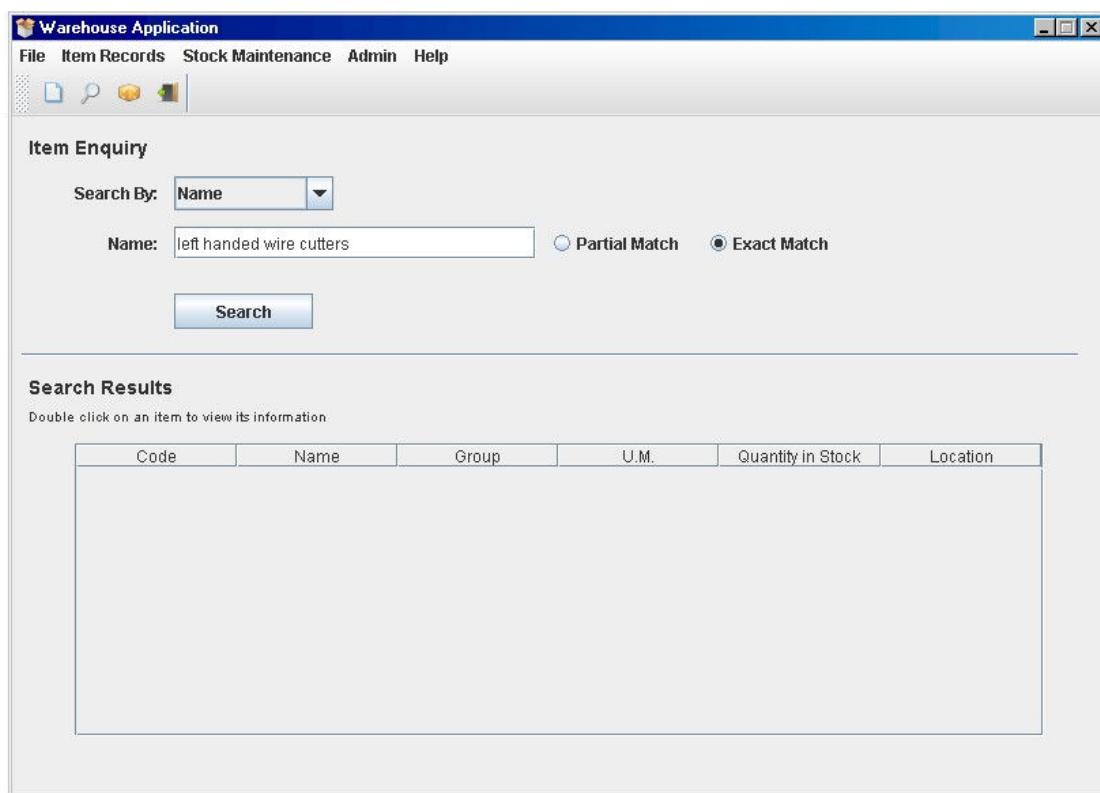
*Test*

Search by: Name

Name: left handed wire cutters

[the search query]

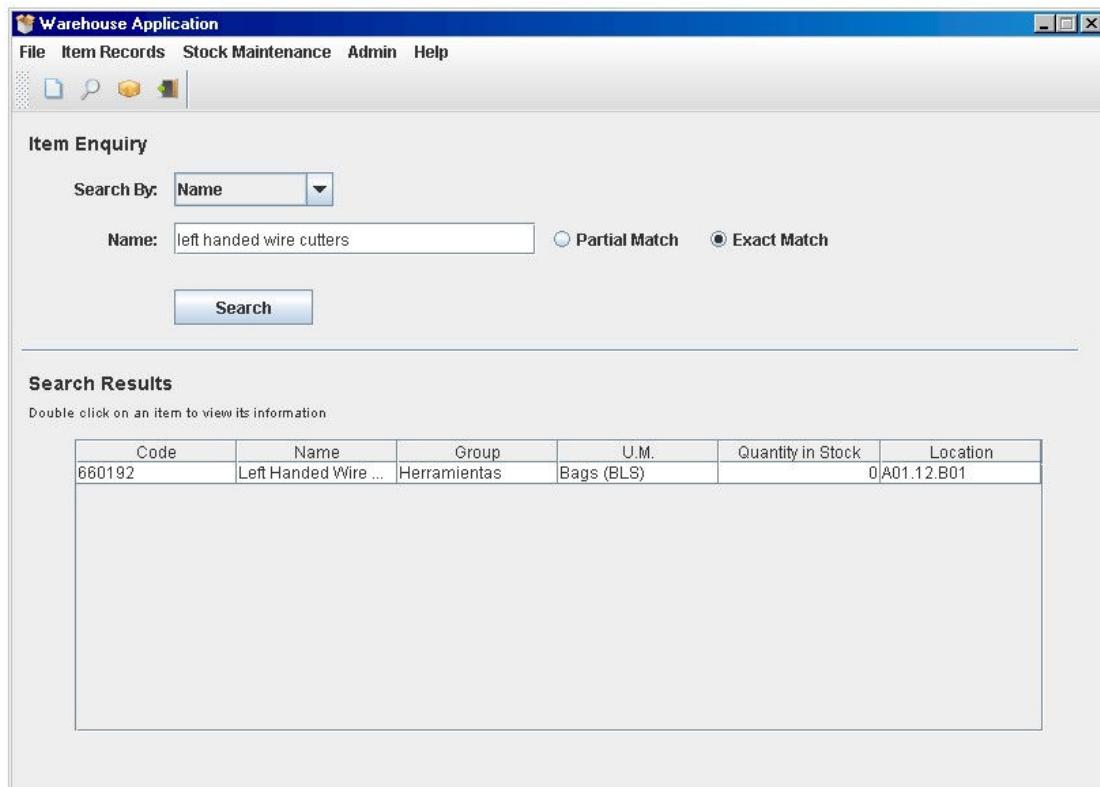
Type: Exact Match



The 'Search' button is pressed

*Results*

Appropriate result is displayed. Note that the search is NOT case sensitive

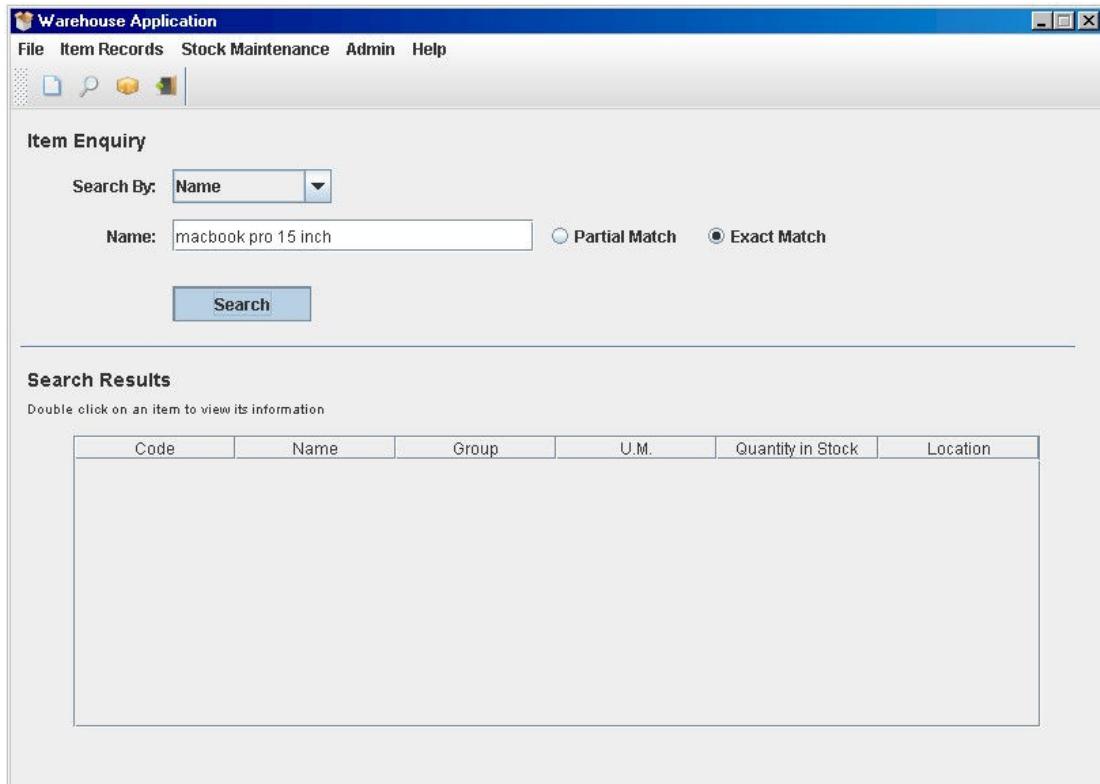


## 2. Searching for an item that does not exist

*Test*

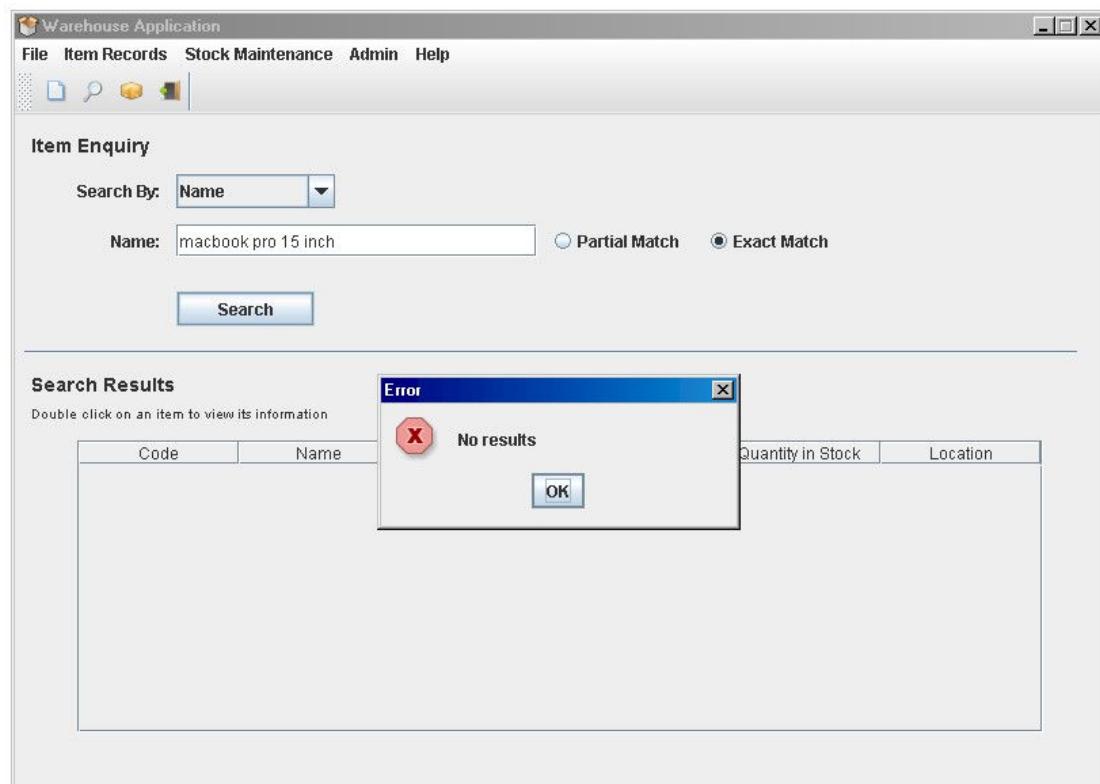
Search by: Name  
Name: macbook pro 15 inch [the search query]  
Type: Exact Match

The 'Search' button is pressed



### Result

The user is informed that no items have been found



## **INCORRECT INPUT**

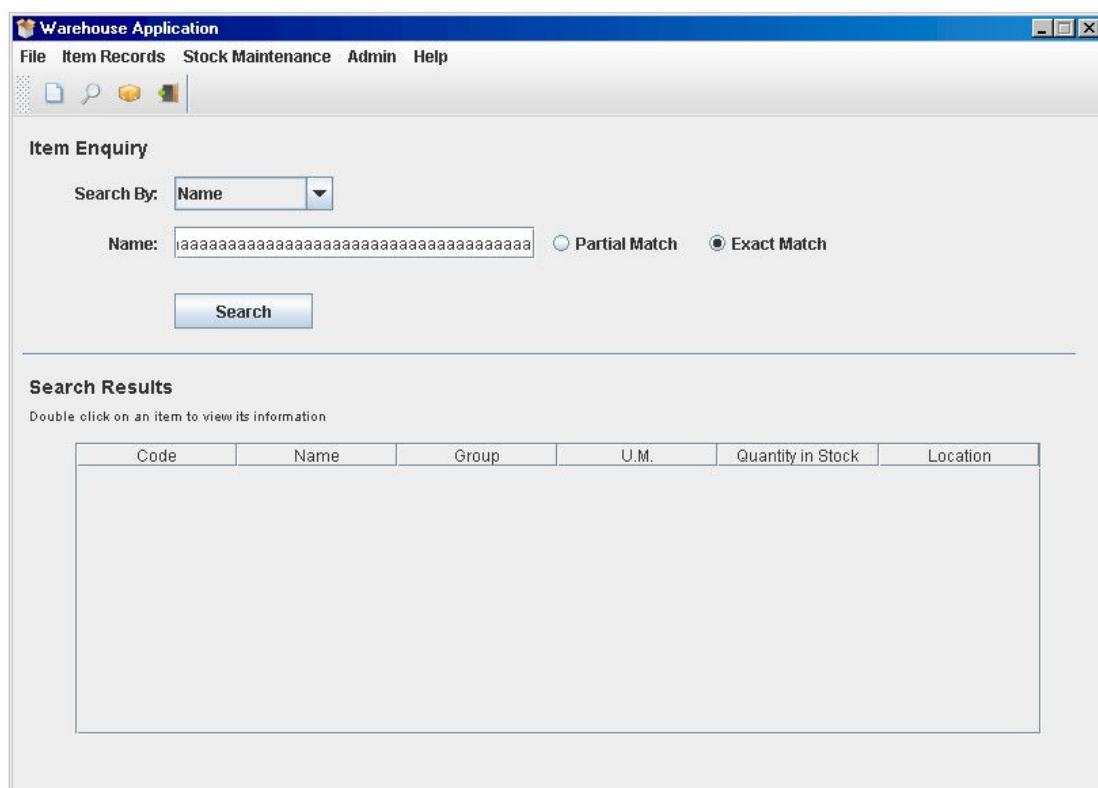
1. A query more than 200 characters in length

Test:

Search by: Name  
Name:  
aa  
aa  
aa  
aa [210 characters]  
Type: Exact Match

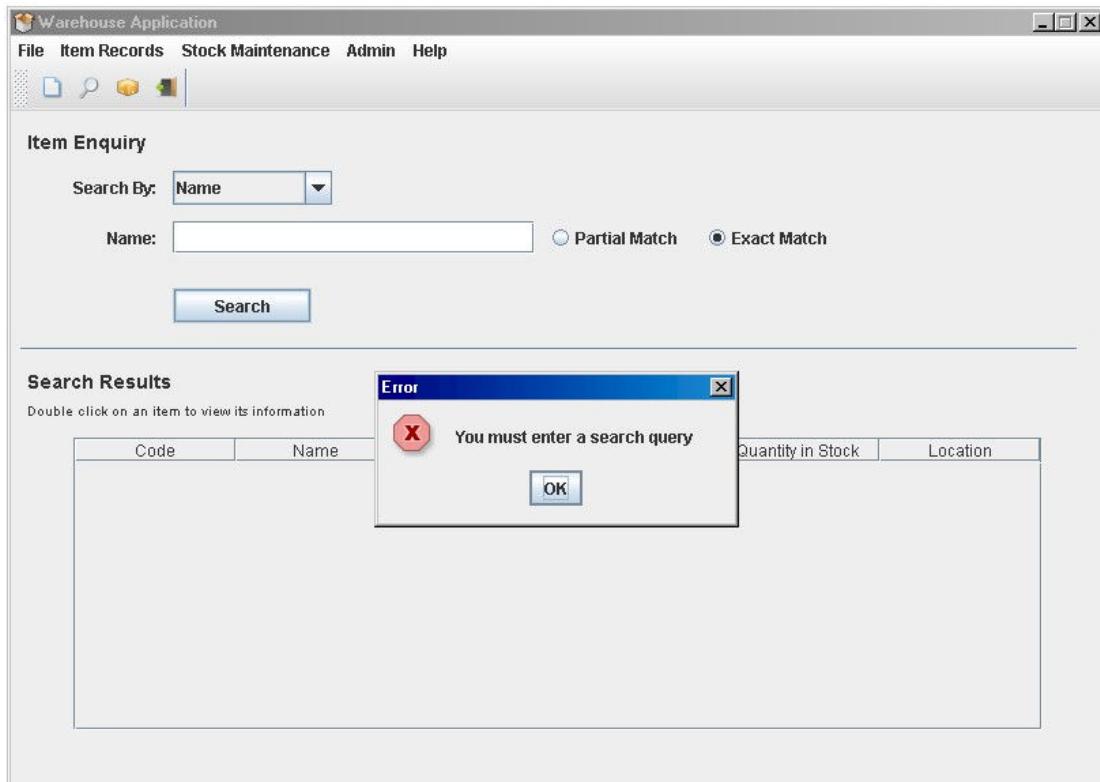
*Result*

The query truncates at 200 characters



3. No (i.e. empty) search query

*Result*



## SEARCHING FOR AN ITEM BY GROUP

Select Group from the drop-down list and the group search boxes load

### CORRECT INPUT

1. Searching for an item that exists

*Test*

Search by:    Group  
Group:        Aceros y Fierros

The screenshot shows the 'Warehouse Application' window with a blue title bar. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for file operations. The main area is titled 'Item Enquiry'. It contains two dropdown menus: 'Search By:' set to 'Group' and 'Group:' set to 'Aceros y fierros'. A 'Search' button is below these fields. A horizontal line separates this from the 'Search Results' section. The 'Search Results' section has a header with columns: 'Code', 'Name', 'Group', 'U.M.', 'Quantity in Stock', and 'Location'. A note below the header says 'Double click on an item to view its information'. The results table is currently empty.

*Result*

The appropriate results are displayed.

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

Item Enquiry

Search By: Group

Group: Aceros y fierros

Search

---

**Search Results**

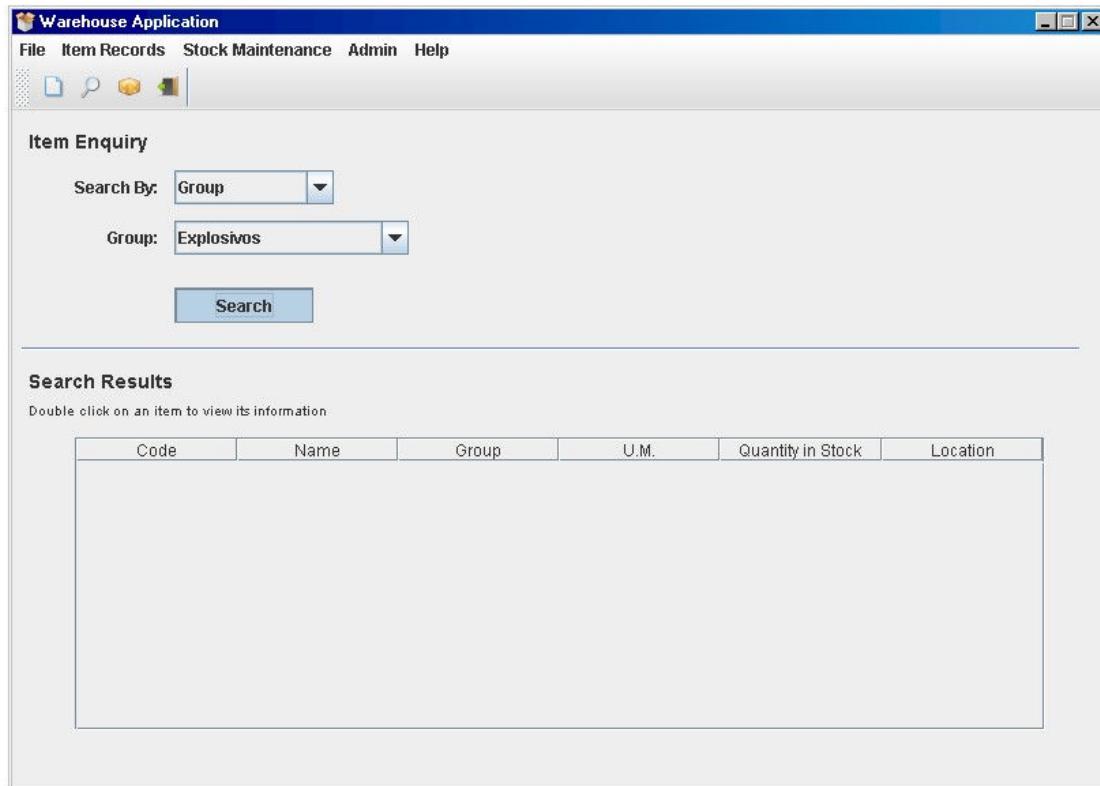
Double click on an item to view its information

Code	Name	Group	U.M.	Quantity in Stock	Location
709125	Steel Nails in 1/16"	Aceros y fierros	Thousands (MLL)	0	A01.28.A05
559207	1/3" Iron Bolts	Aceros y fierros	Hundreds (CIEN)	0	A01.23.E08

2. Searching for a group that contains no items

*Test*

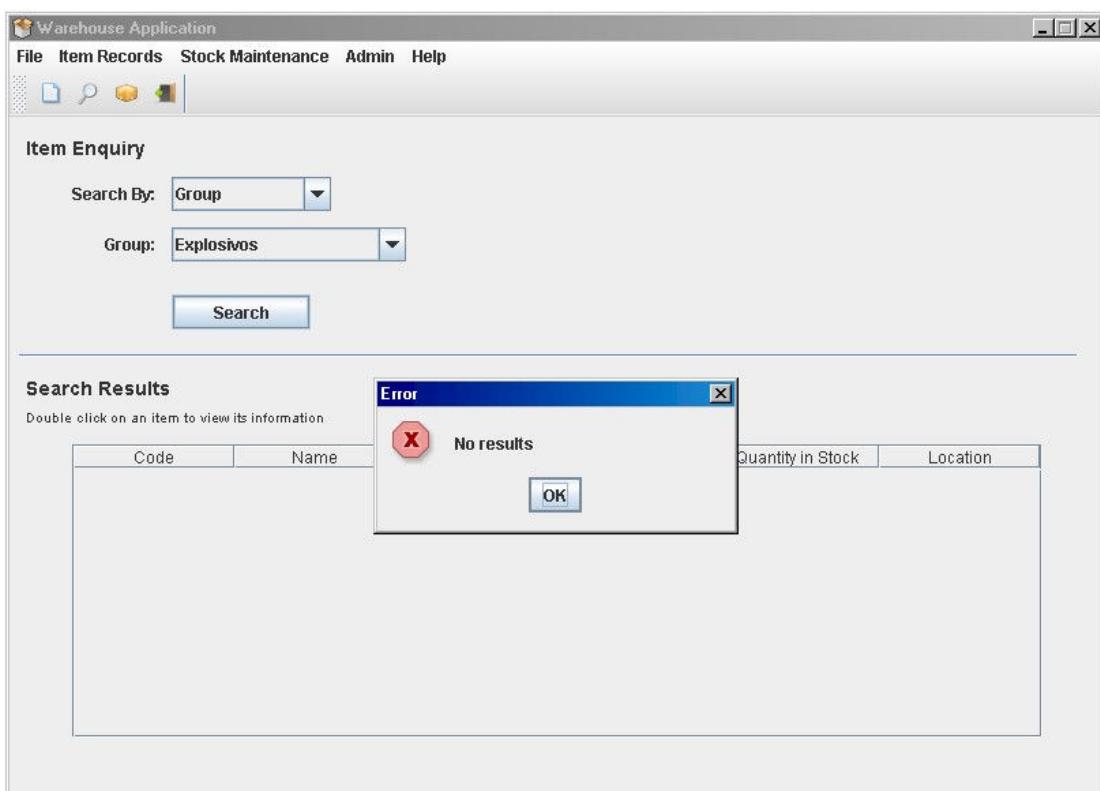
Search by: Group  
Group: Explosivos



Click 'Search' button

*Result*

The user is alerted that the query has yielded no results



## SEARCHING FOR AN ITEM BY LOCATION

Select Location from the drop-down list and the location search boxes load

1. Searching for a location that contains items

*Test*

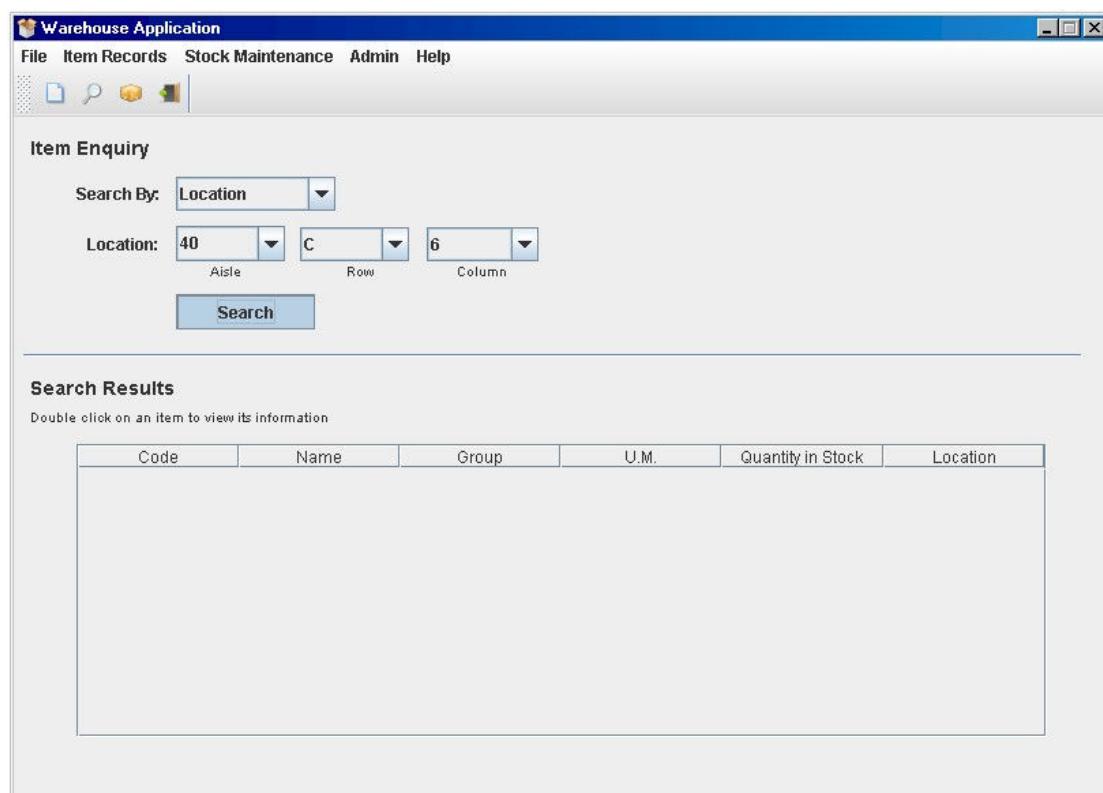
Search by: Location

Location:

Aisle: 40

Row: C

Column: 6



The 'Search' button is clicked

*Result*

Appropriate results are displayed

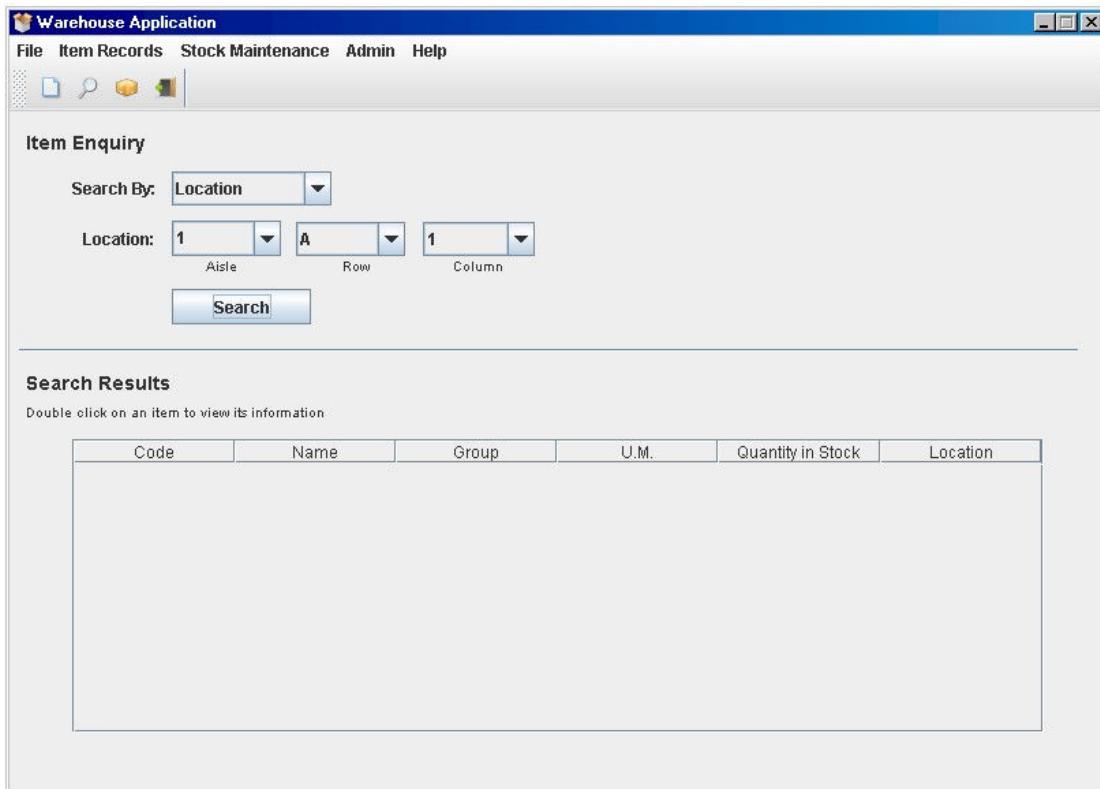
The screenshot shows the 'Warehouse Application' window with the title 'Item Enquiry'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for search, print, and other functions. The main area has a heading 'Item Enquiry' and a search section labeled 'Search By: Location'. Under 'Location', fields show 'Aisle: 40', 'Row: C', and 'Column: 6'. A 'Search' button is present. Below this is a 'Search Results' section with a note 'Double click on an item to view its information'. A table lists items found at the specified location:

Code	Name	Group	U.M.	Quantity in Stock	Location
409821	Synthetic Glue for ...	Varios	Bags (BLS)	0	A01.40.C06
301298	1/2" inch wide nyl... Varios		Bags (BLS)	0	A01.40.C06

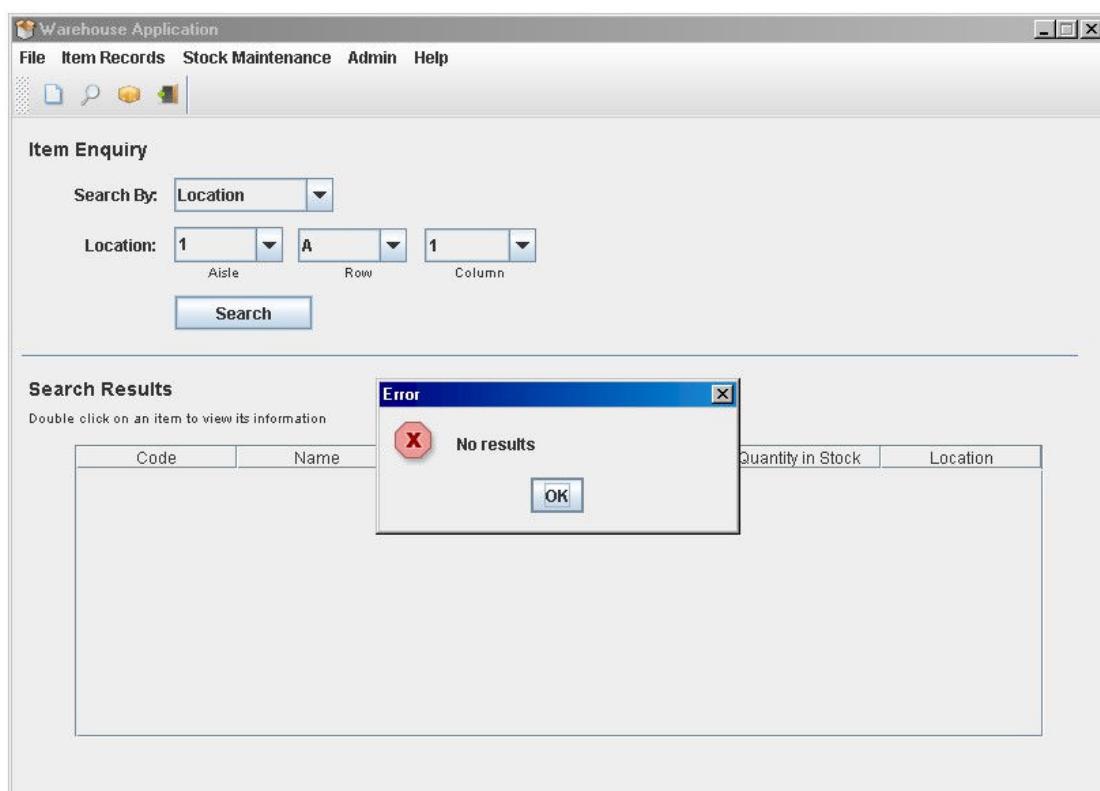
## 2. Searching for a location that contains no items

*Test*

Search by: Location  
Location:  
Aisle: 1  
Row: A  
Column: 1

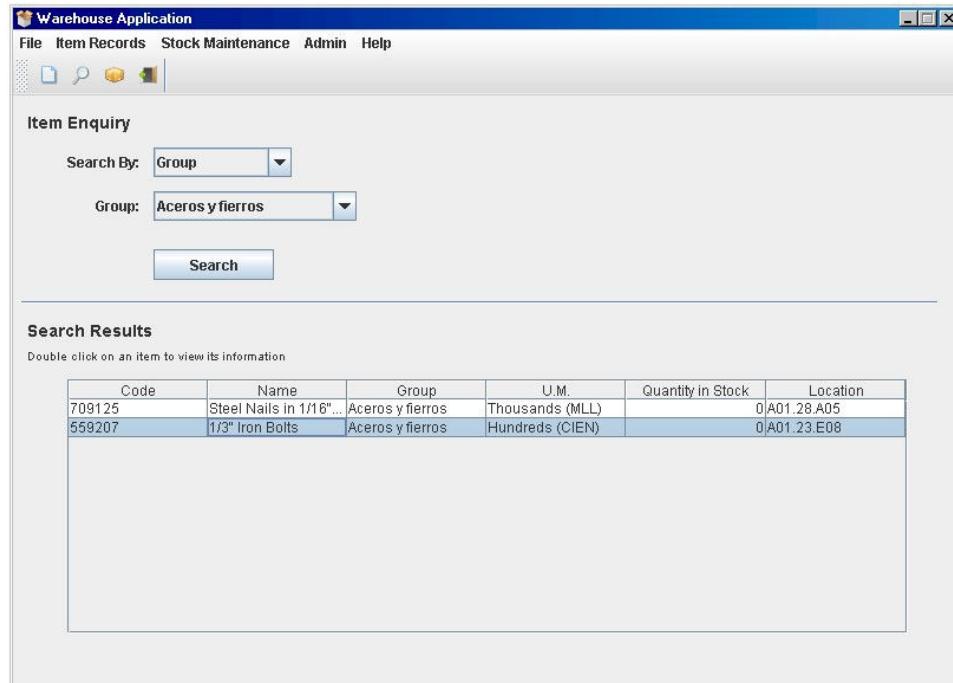


The 'Search' button is clicked



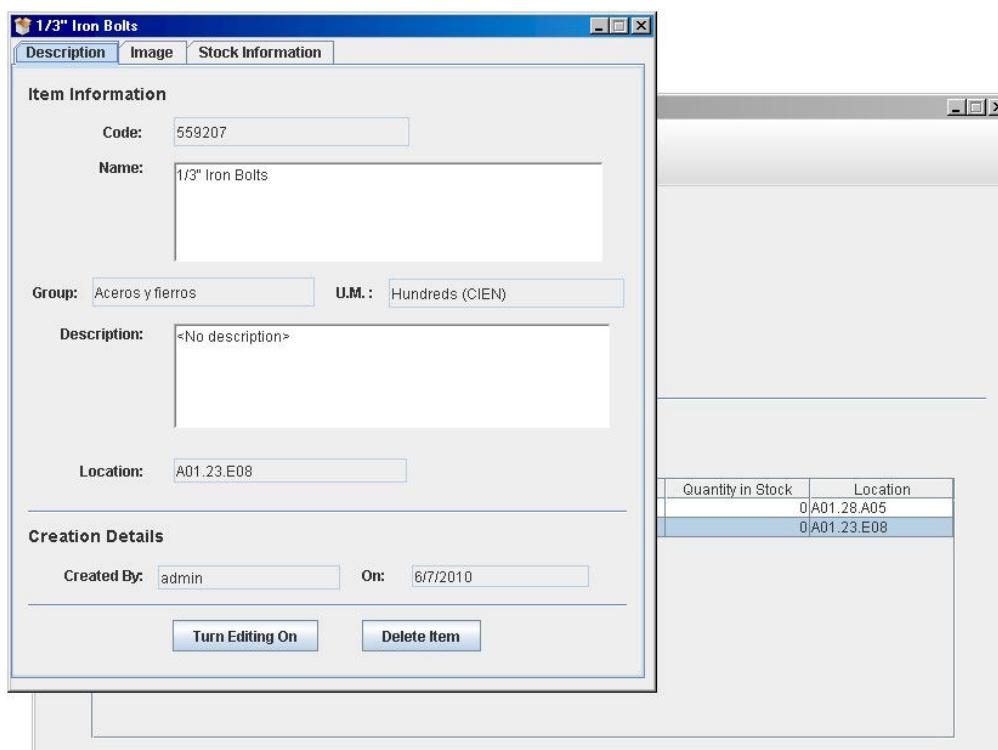
## LOADING THE ITEM DESCRIPTION SCREEN

Double click a search result in the Item Search screen to display the Item Description screen



### Result

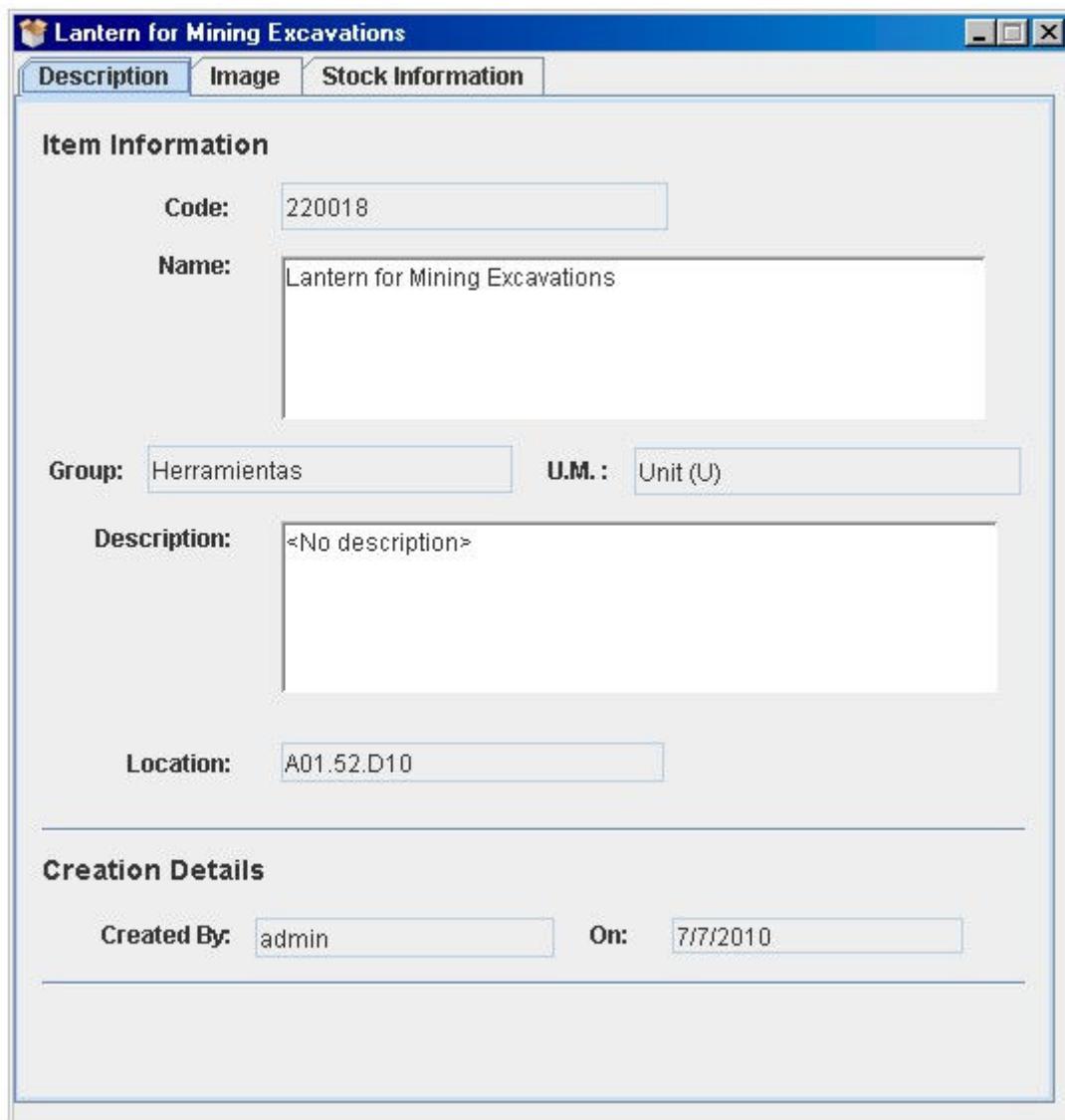
The Item Description screen loads with the item's information



## EDITING ITEM INFORMATION

From the Item Description screen, in 'Description' tab

Users without item entry or administrator permissions have no access to this function



Users with item entry or administrator permissions can see the 'Turn Editing On' button

Lantern for Mining Excavations

Description    Image    Stock Information

**Item Information**

**Code:** 220018

**Name:** Lantern for Mining Excavations

**Group:** Herramientas    **U.M.:** Unit (U)

**Description:** <No description>

**Location:** A01.52.D10

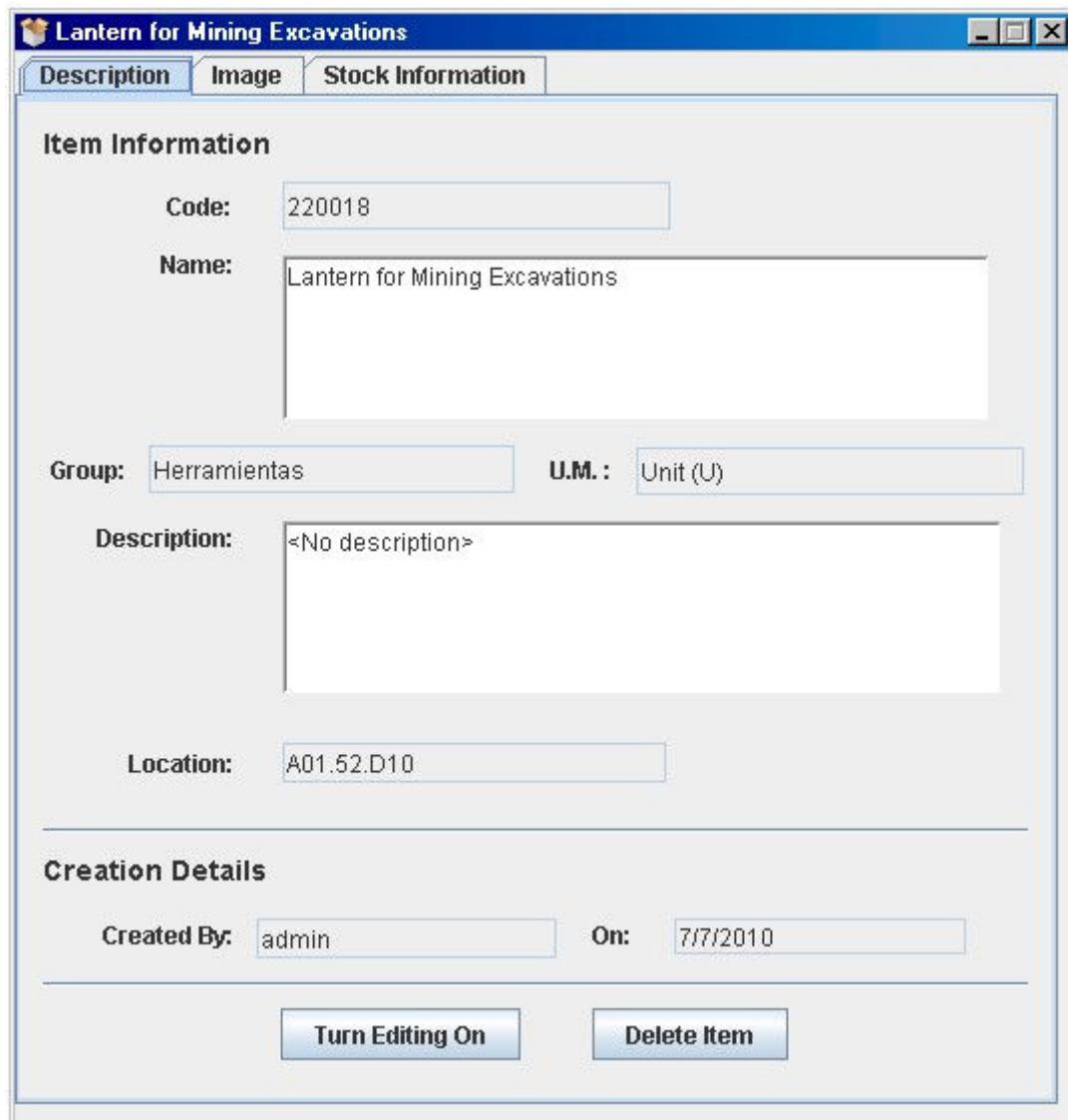
---

**Creation Details**

**Created By:** admin    **On:** 7/7/2010

---

**Turn Editing On**    **Delete Item**



The 'Turn Editing On' button is pressed

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Information**

**Code:** 013294

**Name:** Laptop IBM ThinkPad R52 for Administrative Use

**Group:** Electronico    **U.M.:** Unit (U)

**Description:** 15.4" LCD screen, 1.5 GHz Intel Celeron Processor

**Location:** A01.06.B06

---

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

---

**Turn Editing On**    **Delete Item**

*Result*

The relevant fields activate for input

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Information**

**Code:** 013294 Required. Must be exactly 6 numbers

**Name:** Laptop IBM ThinkPad R52 for Administrative Use  
Required  
Max. 200 characters

**Group:** Electronico    **U.M.:** Unit (U)

**Description:** 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Optional  
Max. 200 characters

**Location:** Aisle: 6    Row: B    Column: 6

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

**Save Changes**    **Delete Item**

### CORRECT INPUT

The item's information will be changed

From:

Code: 013294  
Name: Laptop IBM ThinkPad R52 for Administrative Use  
Group: Electronico  
U.M.: Unit(U)  
Description: 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Location: A01.06.B06

To: [field names in italics are to be changed]

Code: 000000  
Name: Laptop Lenovo ThinkPad R48 for General Use  
Group: Electrico

U.M.: Unit(U)  
Description: 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino Processor  
Location: A01.06.B06

Index by Code file BEFORE item update

Item\_Code.txt - Notepad

File Edit Format View Help

013294	0
220018	1
709125	2
559207	3
409621	4
301298	5
660192	6

Index by Name file BEFORE item update

Item\_Name.txt - Notepad

File Edit Format View Help

Laptop IBM ThinkPad R52 for Administrative Use	0
Lantern for Mining Excavations	1
Steel Nails in 1/16" width	2
1/3" Iron Bolts	3
Synthetic Glue for wood sheets	4
1/2" inch wide nylon rope	5
Left Handed Wire Cutters	6

The 'Save Changes button' is pressed

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Information**

Code: 000000 Required. Must be exactly 6 numbers

Name: Laptop Lenovo ThinkPad R48 for General Use  
Required Max. 200 characters

Group: Electroic U.M.: Unit (U)

Description: 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino Processor  
Optional Max. 200 characters

Location: Aisle 6 Row B Column 6

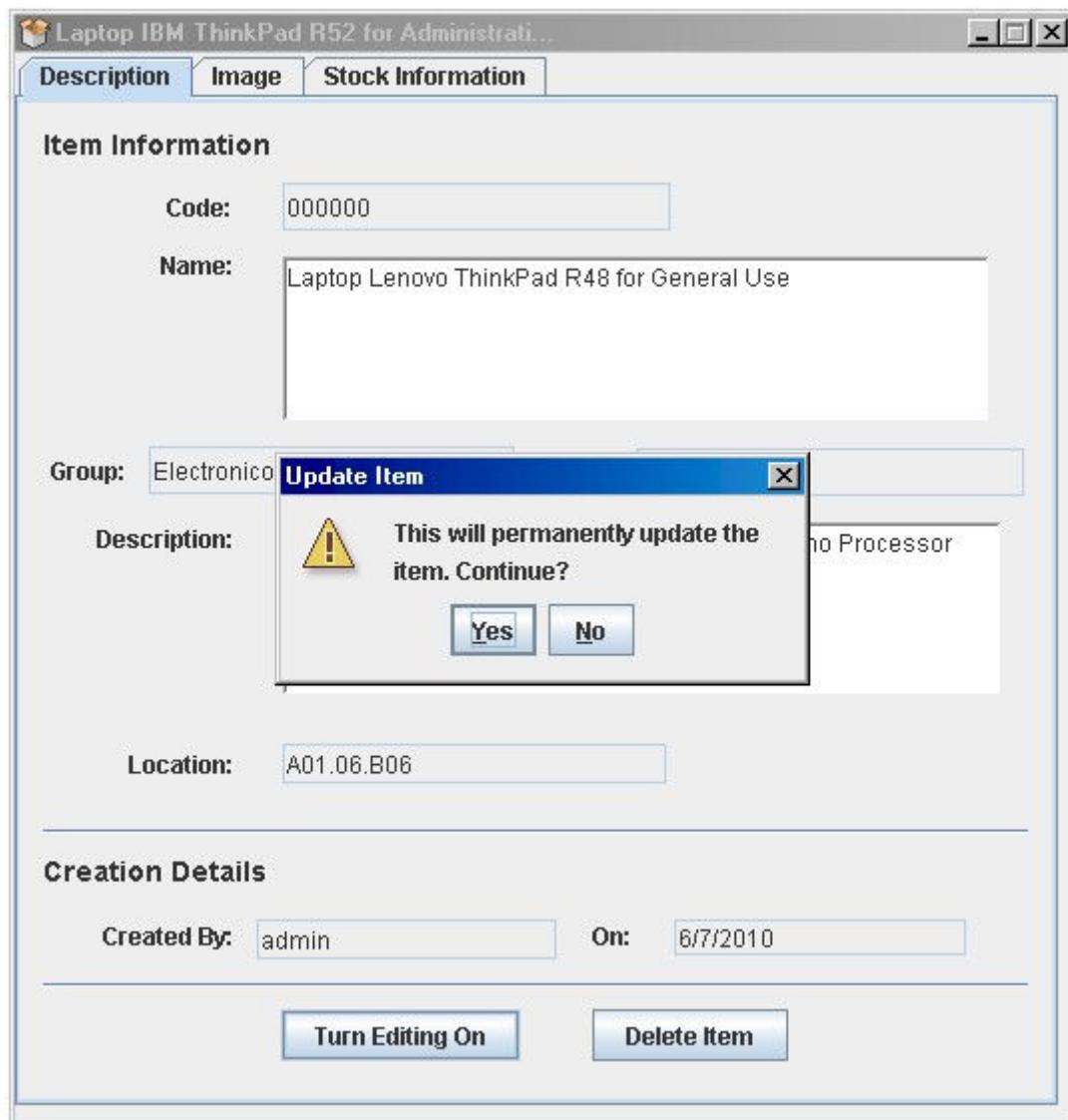
**Creation Details**

Created By: admin On: 6/7/2010

Save Changes Delete Item

*Result*

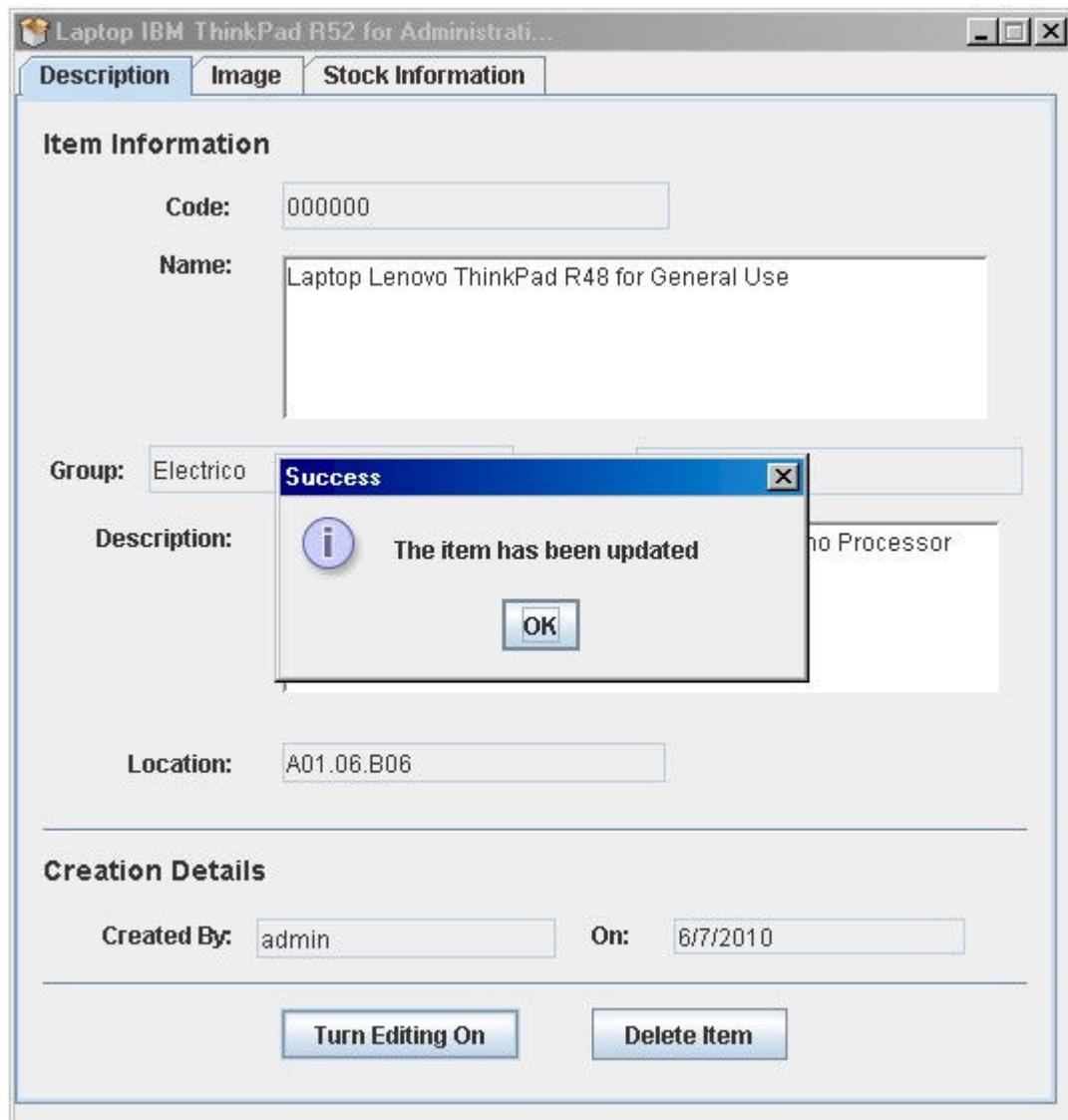
The user is prompted to continue with the edit operation



EITHER Click 'Yes'

*Result*

User informed that edit operation has followed through



Fields are reverted to non-editable mode and the information is updated.

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Information**

**Code:** 000000

**Name:** Laptop Lenovo ThinkPad R48 for General Use

**Group:** Electrico      **U.M.:** Unit (U)

**Description:** 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino Processor

**Location:** A01.06.B06

---

**Creation Details**

**Created By:** admin      **On:** 8/7/2010

**Turn Editing On** **Delete Item**

Index by Code file AFTER item update

Item\_Code.txt - Notepad

File Edit Format View Help

```
000000|0
220018|1
709125|2
559207|3
409621|4
301298|5
660192|6
```

Index by Name file AFTER item update

Item\_Name.txt - Notepad

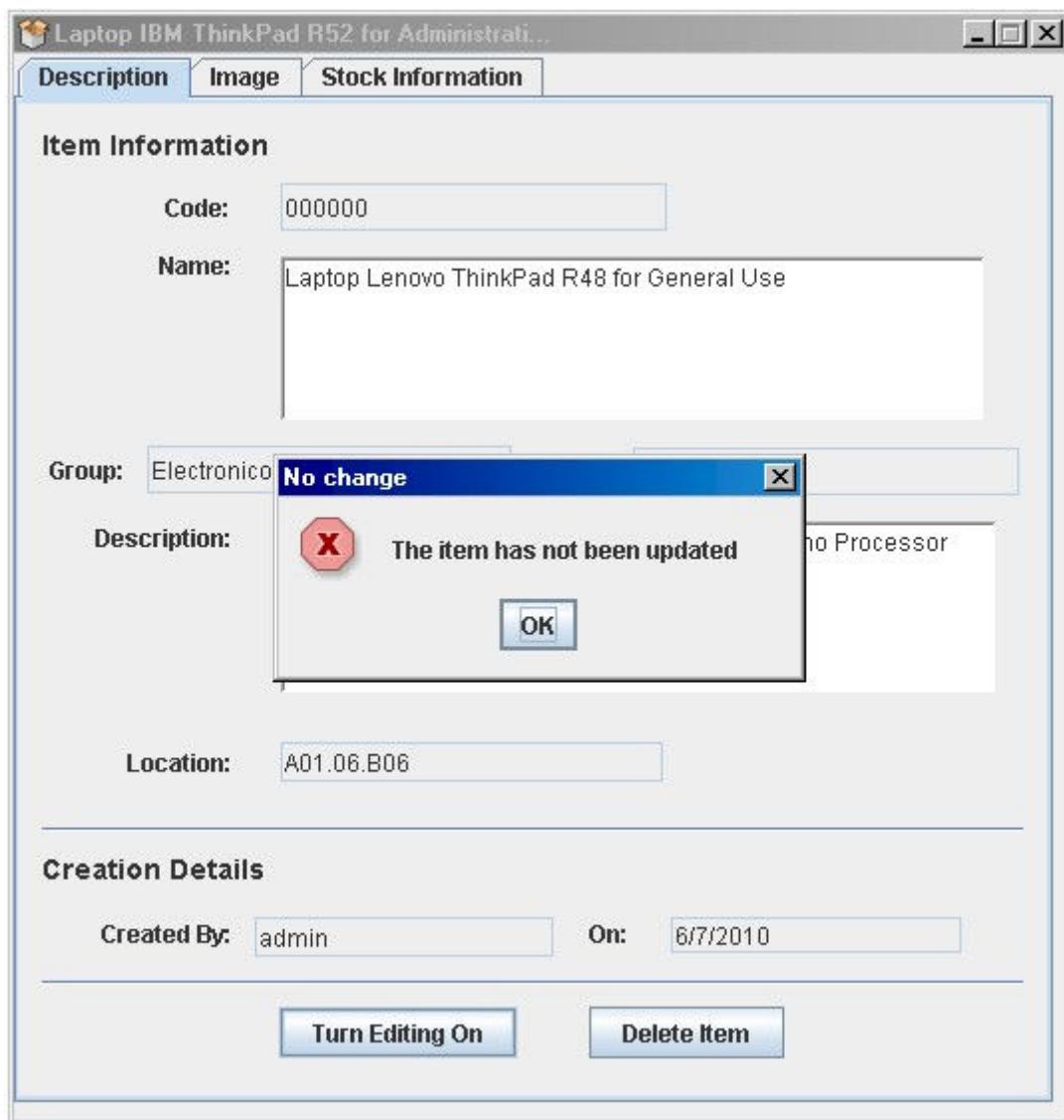
File Edit Format View Help

```
Laptop Lenovo ThinkPad R48 for General Use|0
Lantern for Mining Excavations|1
Steel Nails in 1/16" width|2
1/3" Iron Bolts|3
Synthetic Glue for wood sheets|4
1/2" inch wide nylon rope|5
Left Handed Wire Cutters|6
```

OR Click 'No'

*Result*

User informed that edit operation has not followed through



The 'OK' button is pressed

*Result*

Item information is displayed and fields are reverted to non-editing mode

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Information**

**Code:** 000000

**Name:** Laptop Lenovo ThinkPad R48 for General Use

**Group:** Electronico    **U.M.:** Unit (U)

**Description:** 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino Processor

**Location:** A01.06.B06

---

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

---

**Turn Editing On**    **Delete Item**

INCORRECT INPUT

1. Code containing non-digits
- Name longer than 200 characters
- Description longer than 200 characters

From:

Code: 013294  
Name: Laptop IBM ThinkPad R52 for Administrative Use  
Group: Electronico  
U.M.: Unit(U)  
Description: 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Location: A01.06.B06

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Information**

**Code:** 013294 Required. Must be exactly 6 numbers

**Name:** Laptop IBM ThinkPad R52 for Administrative Use  
Required  
Max. 200 characters

**Group:** Electronico    **U.M.:** Unit (U)

**Description:** 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Optional  
Max. 200 characters

**Location:** Aisle 6, Row B, Column 6

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

**Save Changes**    **Delete Item**

To: [field names in italics are to be changed]

**Code:** abc123  
**Name:**aa  
aa  
aa  
aa  
aa  
aa  
[250 characters]  
**Group:** Electronico  
**U.M.:** Unit(U)  
**Description:** 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino  
Processor  
**Location:** A01.06.B06

*Result*

Code field only allows digits, the name field is truncated to the maximum length

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Information**

**Code:** 123 Required. Must be exactly 6 numbers

**Name:**  
Required  
Max. 200 characters  
aa

**Group:** Electronico    **U.M.:** Unit (U)

**Description:**  
Optional  
Max. 200 characters  
15.4" LCD screen, 1.5 GHz Intel Celeron Processor

**Location:** Aisle Row Column  
6 B 6

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

**Save Changes** **Delete Item**

### INCORRECT INPUT

1. One or more required fields missing, code under 6 digits

From:

Code: 013294  
Name: Laptop IBM ThinkPad R52 for Administrative Use  
Group: Electronico  
U.M.: Unit(U)  
Description: 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Location: A01.06.B06

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Information**

**Code:** 013294 Required. Must be exactly 6 numbers

**Name:** Laptop IBM ThinkPad R52 for Administrative Use  
Required  
Max. 200 characters

**Group:** Electronico    **U.M.:** Unit (U)

**Description:** 15.4" LCD screen, 1.5 GHz Intel Celeron Processor  
Optional  
Max. 200 characters

**Location:** Aisle: 6    Row: B    Column: 6

**Creation Details**

**Created By:** admin    **On:** 6/7/2010

**Save Changes**    **Delete Item**

To: [field names in italics are to be changed]

Code: 00 [2 digits]  
Name: [empty]  
Group: Electronico  
U.M.: Unit(U)  
Description: 13.1" diagonal LCD screen, 1.5 GHz Intel Centrino Processor  
Location: A01.06.B06

The 'Save Changes' button is pressed

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Information**

**Code:** 00 Required. Must be exactly 6 numbers

**Name:**  
Required  
Max. 200 characters

**Group:** Electronico **U.M.:** Unit (U)

**Description:** 15." LCD screen, 1.5 GHz Intel Celeron Processor  
Optional  
Max. 200 characters

**Location:** Aisle Row Column  
6 B 6

**Creation Details**

**Created By:** admin **On:** 7/7/2010

**Save Changes** **Delete Item**

*Result*

User informed that some fields are missing or incorrect

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Information**

**Code:** 00 Required. Must be exactly 6 numbers

**Name:**  
Required  
Max. 200 characters

**Group:** Electronic

**Description:**  
Optional  
Max. 200 characters

**Aisle** 6    **Row** B    **Column** 6

**Creation Details**

**Created By:** admin    **On:** 7/7/2010

**Save Changes** **Delete Item**

An error dialog box is displayed in the center of the form, titled "Error". It contains a red "X" icon and the message "Some fields are missing or incorrect". A blue "OK" button is at the bottom right of the dialog.

## VIEWING ITEM IMAGE

### 1. An item that has an image

The item with the name 'Orange alert vest', whose image was created above, is searched through a partial name search with the query 'orange alert'

The item is found

The screenshot shows a Windows application window titled 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for file operations. The main area is divided into two sections: 'Item Enquiry' and 'Search Results'. In the 'Item Enquiry' section, 'Search By' is set to 'Name', the 'Name' field contains 'orange alert', and the 'Partial Match' radio button is selected. A 'Search' button is present. In the 'Search Results' section, there is a message 'Double click on an item to view its information'. A table displays one item record:

Code	Name	Group	U.M.	Quantity in Stock	Location
490111	Orange alert vests	Seguridad, equipo ...	Unit (U)	0	A01.03.E05

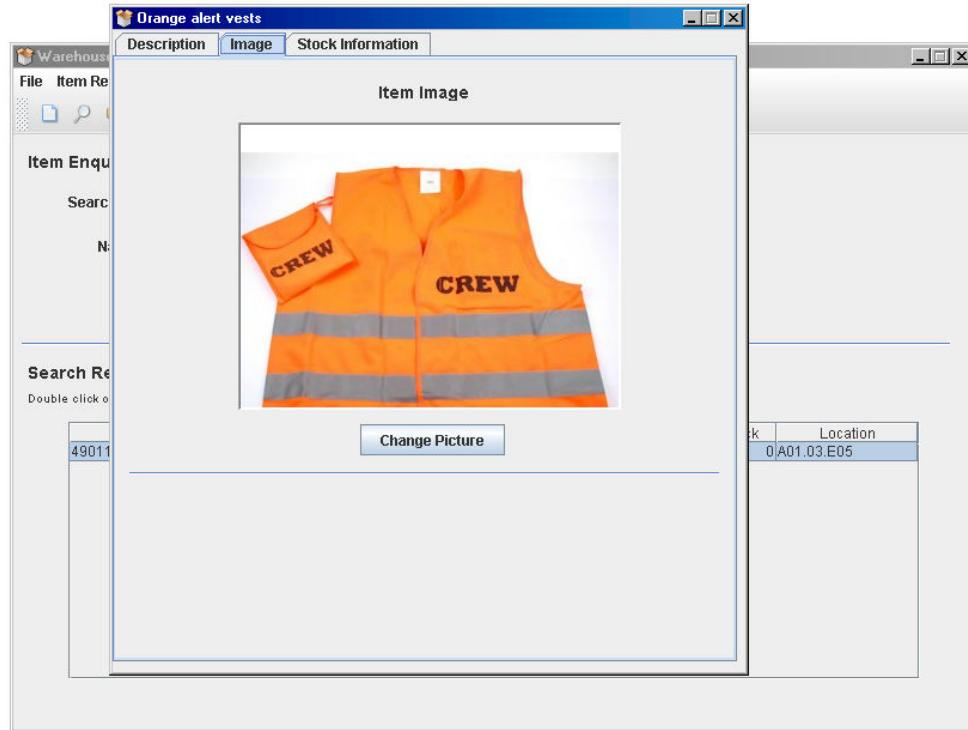
The item is double clicked

The Item Description screen opens

The 'Image' tab is clicked

*Result*

The correct item image is shown.

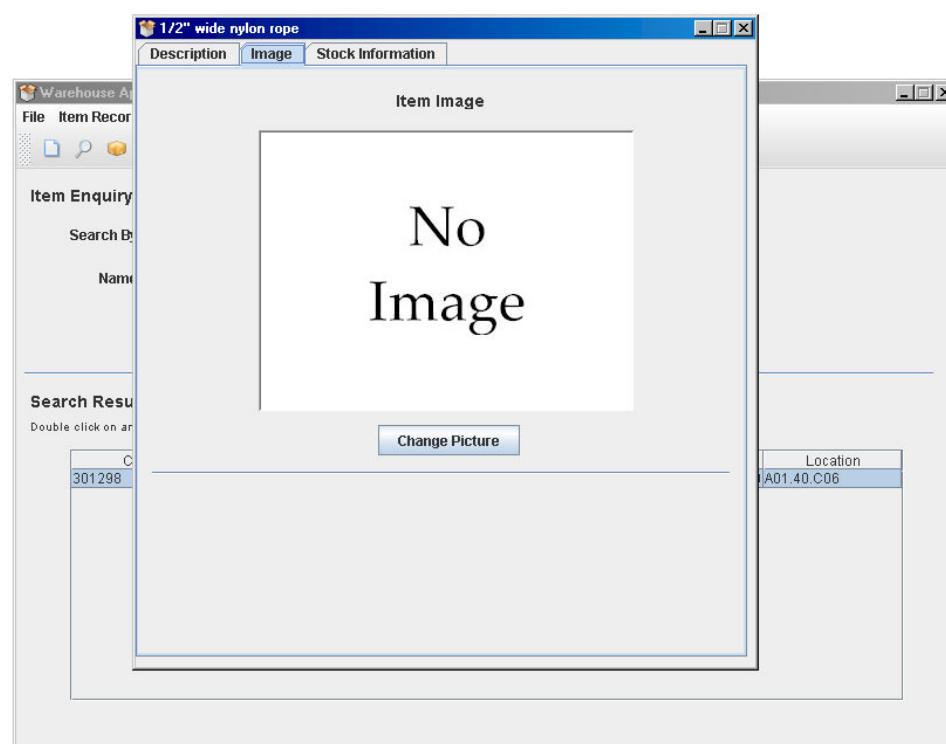


## 2. An item that has no image

The same procedure is repeated for the item with name '1/2" wide nylon rope'

### *Result*

The user is shown that the item has no image

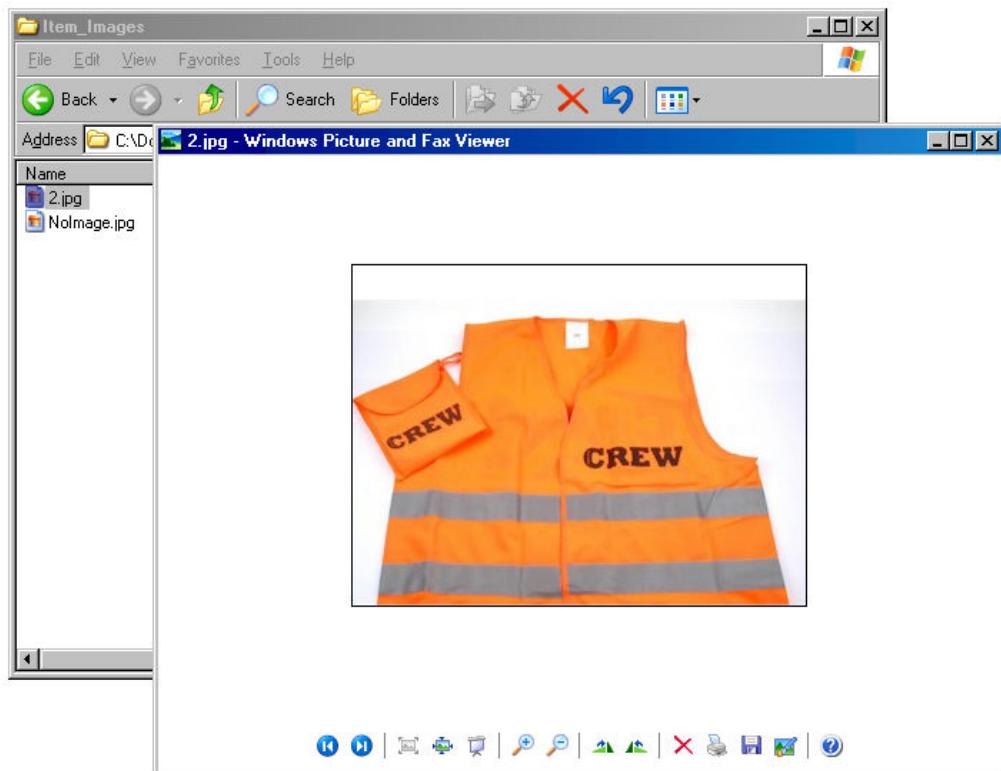


## CHANGING AN ITEM'S IMAGE

The image of the item with name 'Orange alert vests' will be changed.



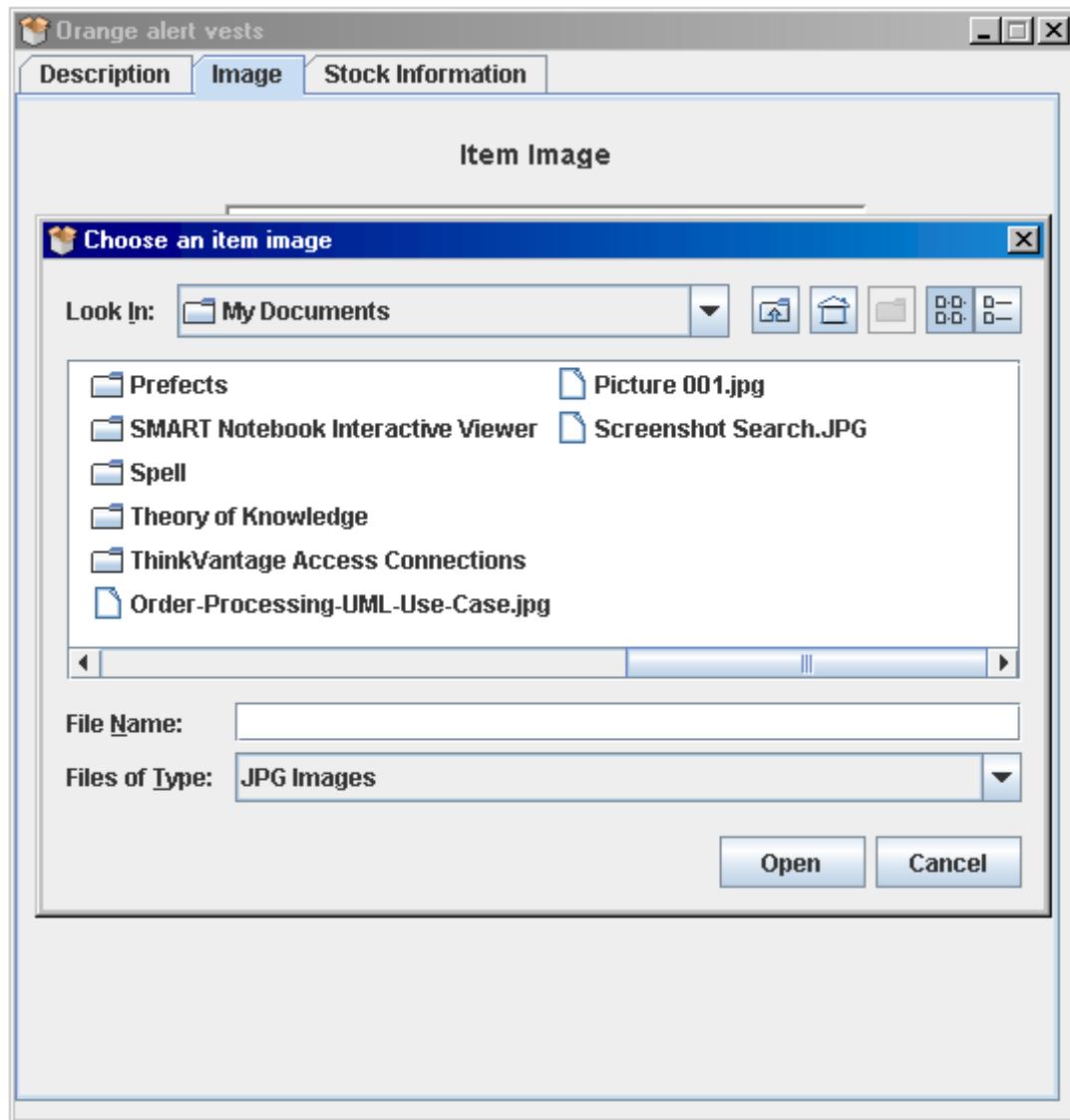
The file BEFORE the image change. Note that its filename is '2.jpg'



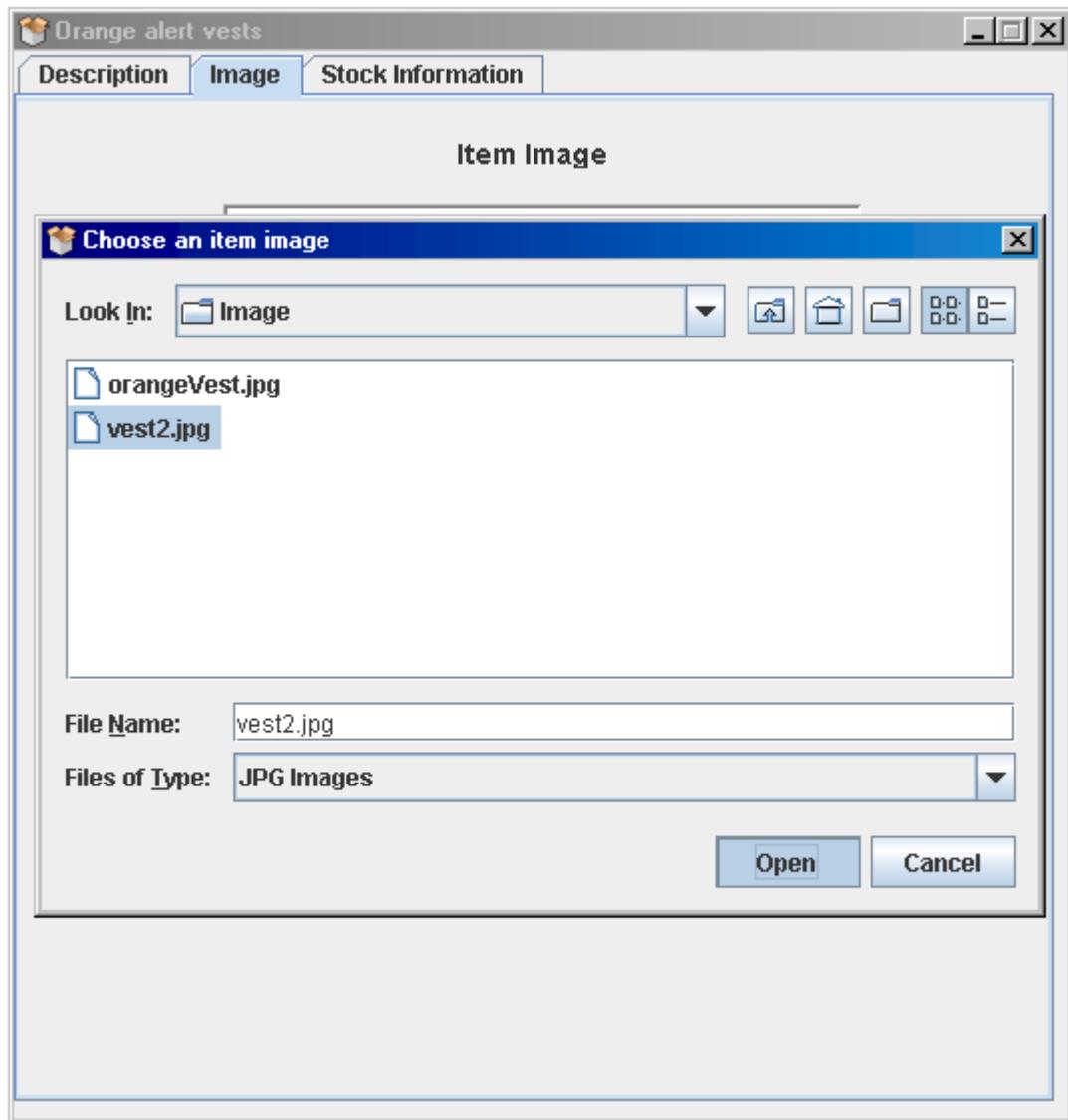
The 'Change Picture' button is pressed

*Result*

The file chooser to change the Item Image opens. Note that only .jpg images are displayed.



The image 'vest2.jpg' in the 'Image' directory is selected and 'Open' is clicked

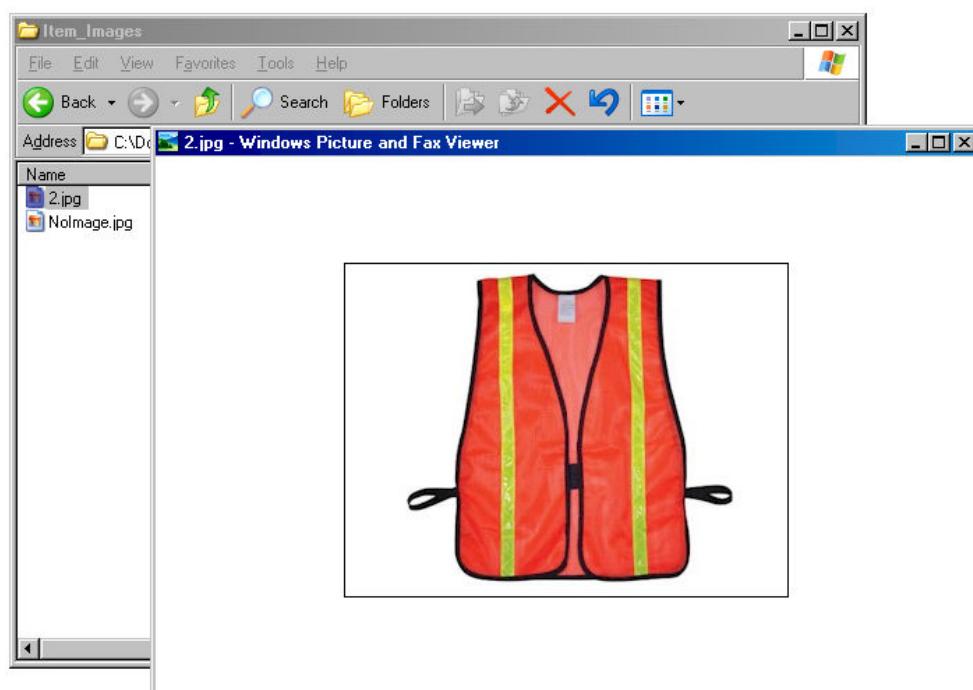


*Result*

The new item image is displayed.



The image is copied to the item images directory and renamed appropriately

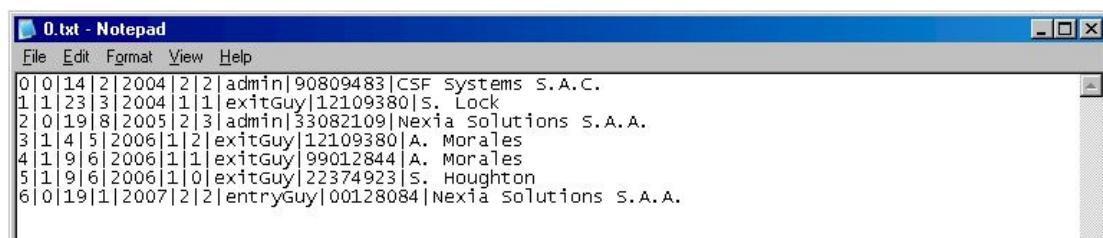


## VIEWING ITEM TRANSACTIONS

From the Item Description screen, 'Stock Information' tab

Viewing the transactions for the item with name 'Laptop IBM ThinkPad R52 for Administrative Use'

This item has ID 0. Transactions file for this item:



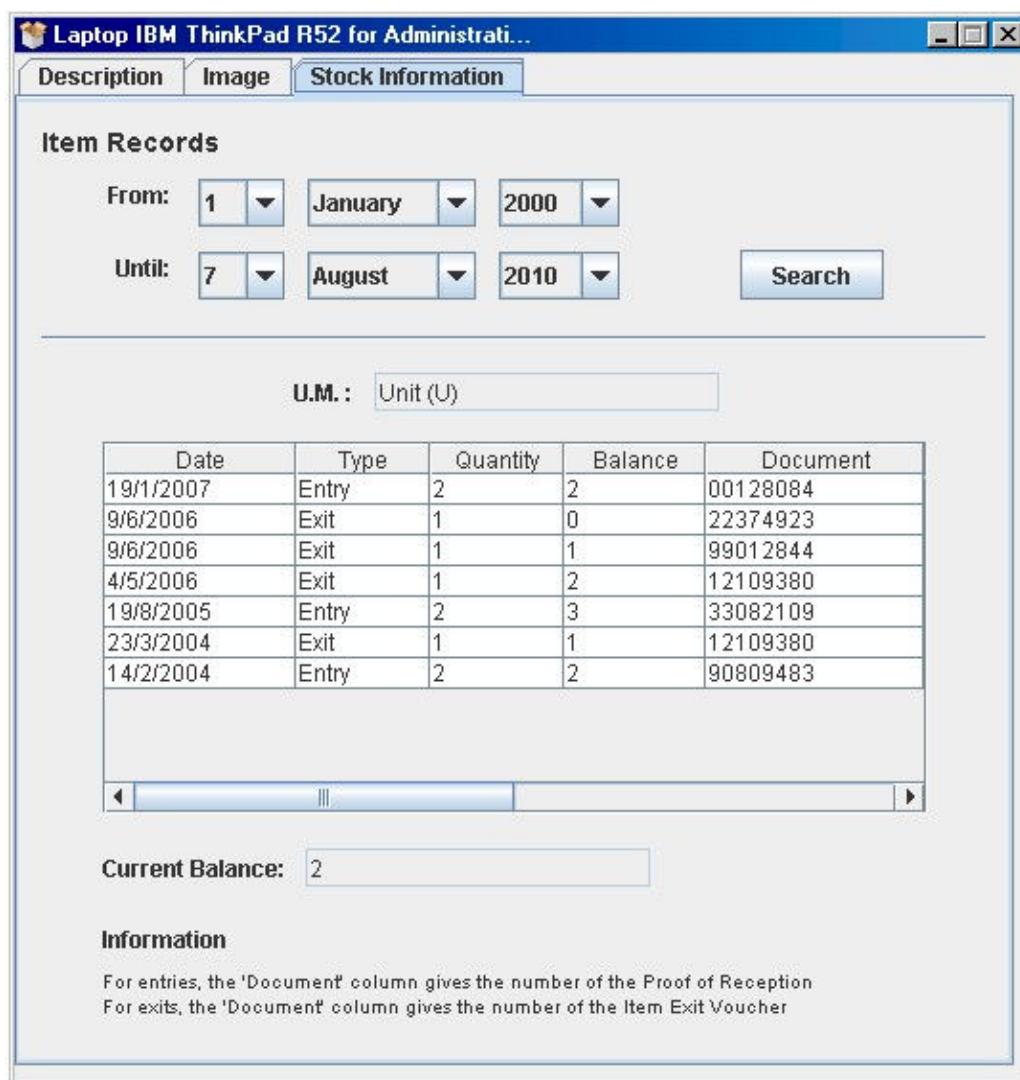
0.txt - Notepad

File Edit Format View Help

```
0|0|14|2|2004|2|2|admin|90809483|CSF Systems S.A.C.  
1|1|23|3|2004|1|1|exitGuy|12109380|S. Lock  
2|0|19|8|2005|2|3|admin|33082109|Nexia Solutions S.A.A.  
3|1|4|5|2006|1|2|exitGuy|12109380|A. Morales  
4|1|9|6|2006|1|1|exitGuy|99012844|A. Morales  
5|1|9|6|2006|1|0|exitGuy|22374923|S. Houghton  
6|0|19|1|2007|2|2|entryGuy|00128084|Nexia Solutions S.A.A.
```

Default view:

All transactions from January 1 2000 to current date display:



Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000

Until: 7 August 2010

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

Current Balance: 2

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

To change the date range of transactions to display:

**CORRECT INPUT**

From: 6 June 2005  
To: 2 March 2008

Click the 'Search' button

The screenshot shows a Windows application window titled "Laptop IBM ThinkPad R52 for Administrati...". The window has a tab bar with "Description", "Image", and "Stock Information" tabs, where "Stock Information" is selected. Below the tabs is a section titled "Item Records" containing two sets of date selection dropdowns: "From" (set to 6 June 2005) and "Until" (set to 2 March 2008), followed by a "Search" button. Below these is a unit selector labeled "U.M.: Unit (U)". A large table displays transaction history with columns: Date, Type, Quantity, Balance, and Document. The table contains the following data:

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

Below the table is a navigation bar with arrows and a page number indicator. At the bottom, there is a "Current Balance" field containing the value "2" and an "Information" section with explanatory text.

*Result*

Correct transactions in the interval display, in descending order by date  
(newest transaction in the first row of the table)

Laptop IBM ThinkPad R52 for Administrati...

Description    Image    Stock Information

**Item Records**

From:  June 2005

Until:  March 2008

**Search**

**U.M.:** Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109

Current Balance:

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

### INCORRECT INPUT

A start date after the end date

From: 1 December 2009  
To: 1 January 2001

Click the 'Search' button

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 December 2009

Until: 1 January 2001

Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109

< ||| >

Current Balance: 2

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

*Result*

User informed that the date range is invalid

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 December 2009

Until: 1 January 2001

U.M.: Unit (U)

Date Document

19/1/2007	4
9/6/2006	3
9/6/2006	4
4/5/2006	0
19/8/2005	9
23/3/2004	0
14/2/2004	90809483

Entry | 2 | 2 | 90809483

Current Balance: 2

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

## DELETING A TRANSACTION

Only accessible to Admin users

A user with no administrator permissions does not see the 'Delete Record' button

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

A user WITH administrator permissions sees the 'Delete Record' button

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000

Until: 7 August 2010

Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

< | | >

Current Balance: 2 Delete Record

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

The delete button is disabled if no row is selected. If a row is selected, it becomes enabled

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000

Until: 7 August 2010

Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

< ||| >

Current Balance: 2 Delete Record

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

Press the 'Delete Record' button when the transaction dated 4/5/2006 is selected

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000

Until: 7 August 2010

Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

< ||| >

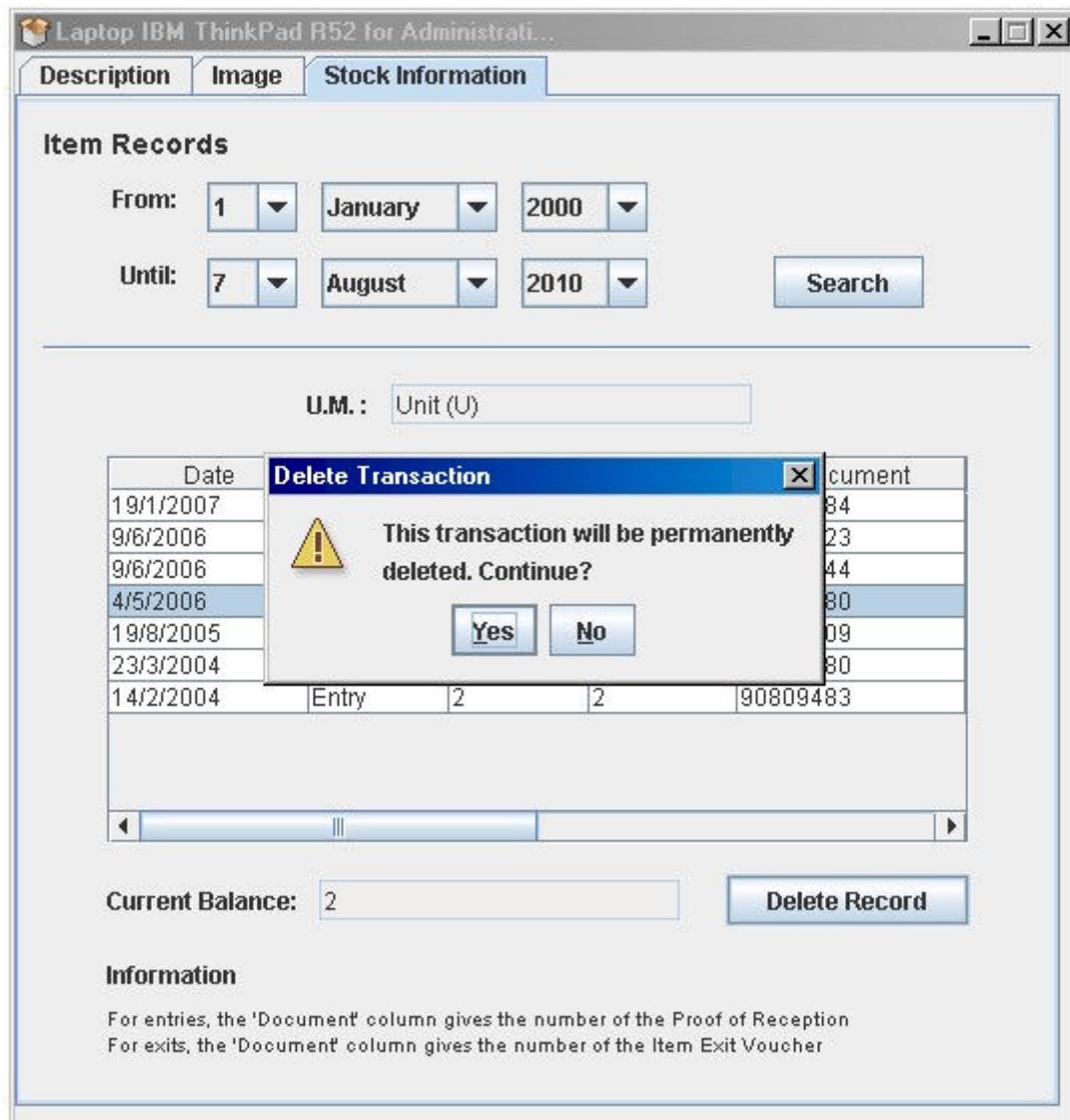
Current Balance: 2 Delete Record

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

*Result*

User prompted for continuation to delete



EITHER click 'Yes'

Transactions file for this item BEFORE deletion

```
File Edit Format View Help
0|0|14|2|2004|2|2|admin|90809483|CSF Systems S.A.C.
1|1|23|3|2004|1|1|exitGuy|12109380|S. Lock
2|0|19|8|2005|2|3|admin|33082109|Nexia Solutions S.A.A.
3|1|4|5|2006|1|2|exitGuy|12109380|A. Morales
4|1|9|6|2006|1|1|exitGuy|99012844|A. Morales
5|1|9|6|2006|1|0|exitGuy|22374923|S. Houghton
6|0|19|1|2007|2|2|entryGuy|00128084|Nexia Solutions S.A.A.
```

*Result*

User informed that the transaction was deleted

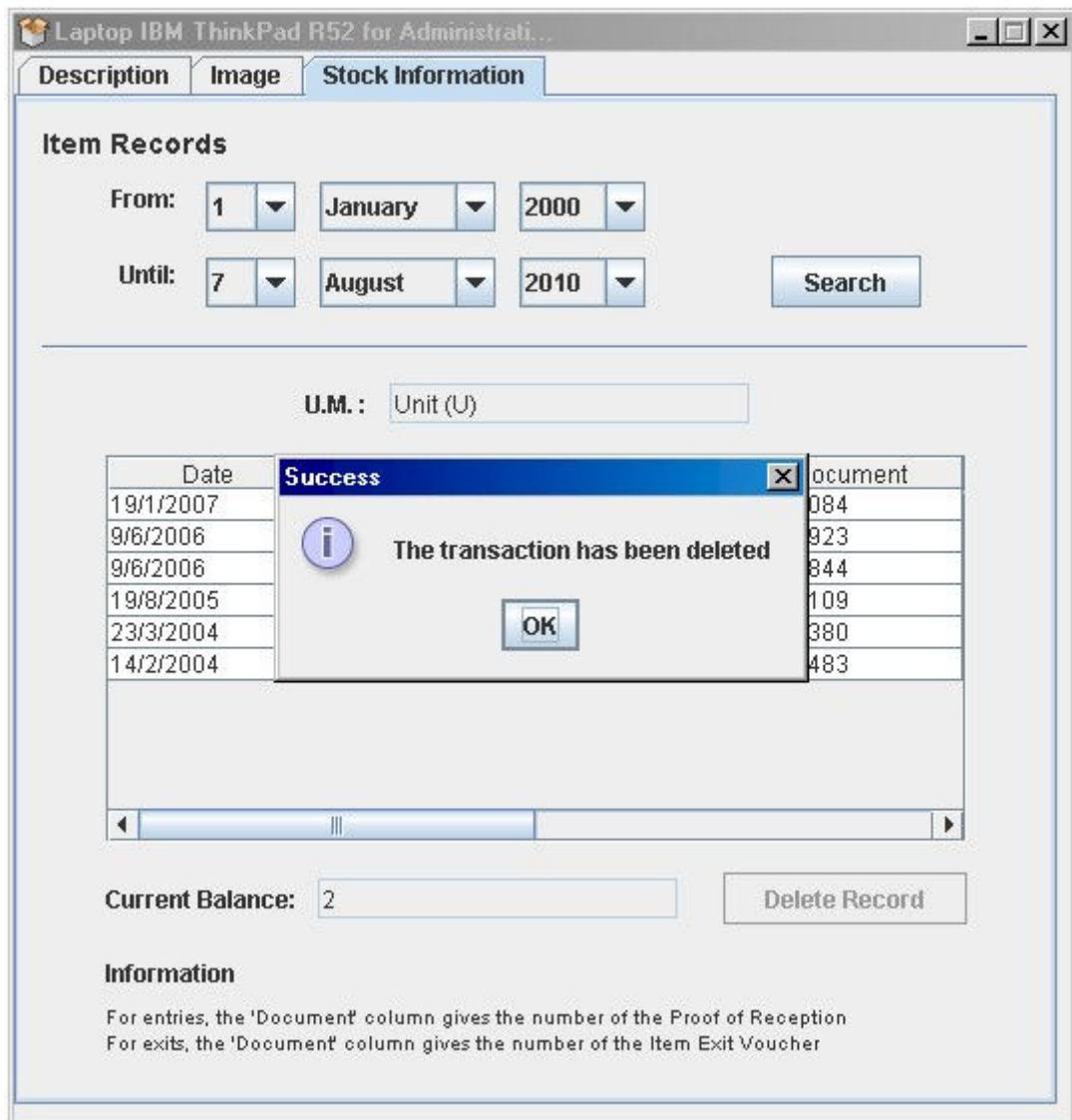


Table updated, does not show deleted transaction

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000 Until: 7 August 2010 Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

< || >

Current Balance: 2 Delete Record

**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

Transactions file for this item AFTER deletion

Transaction deleted from file

0.txt - Notepad

File Edit Format View Help

```
0|0|14|2|2004|2|2|admin|90809483|CSF Systems S.A.C.  
1|1|23|3|2004|1|1|exitGuy|12109380|S. Lock  
2|0|19|8|2005|2|3|admin|33082109|Nexia Solutions S.A.A.  
4|1|9|6|2006|1|1|exitGuy|99012844|A. Morales  
5|1|9|6|2006|1|0|exitGuy|22374923|S. Houghton  
6|0|19|1|2007|2|2|entryGuy|00128084|Nexia solutions S.A.A.
```

OR click 'No'

*Result*

Transaction not deleted

Laptop IBM ThinkPad R52 for Administrati...

Description Image Stock Information

**Item Records**

From: 1 January 2000

Until: 7 August 2010

Search

U.M.: Unit (U)

Date	Type	Quantity	Balance	Document
19/1/2007	Entry	2	2	00128084
9/6/2006	Exit	1	0	22374923
9/6/2006	Exit	1	1	99012844
4/5/2006	Exit	1	2	12109380
19/8/2005	Entry	2	3	33082109
23/3/2004	Exit	1	1	12109380
14/2/2004	Entry	2	2	90809483

< || >

Current Balance: 2

Delete Record

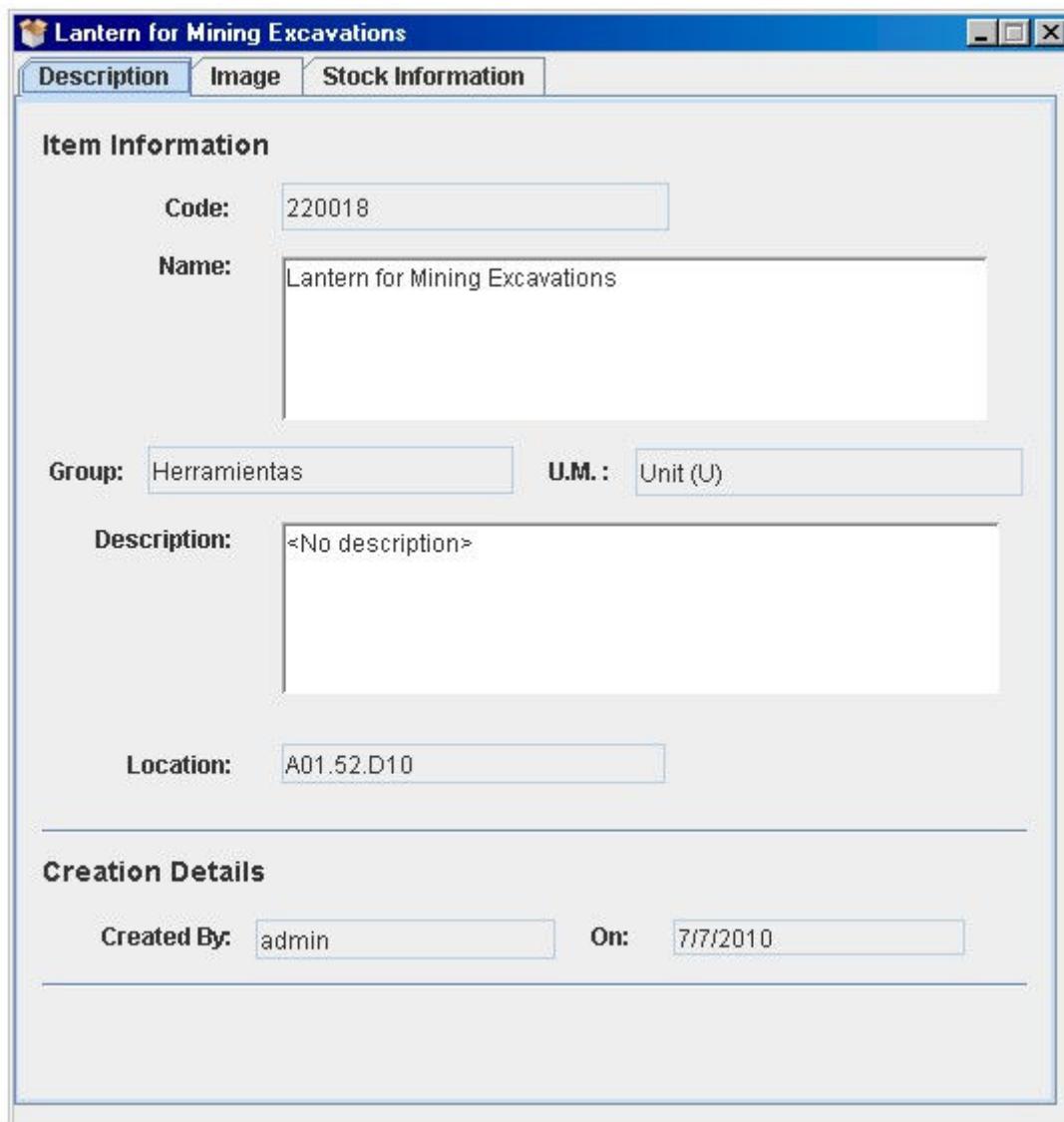
**Information**

For entries, the 'Document' column gives the number of the Proof of Reception  
For exits, the 'Document' column gives the number of the Item Exit Voucher

## DELETING AN ITEM

From the Item Description screen, in 'Description' tab

Users without item entry or administrator permissions have no access to this function



Users with item entry or administrator permissions can see the 'Delete Item' button

Lantern for Mining Excavations

Description    Image    Stock Information

**Item Information**

Code: 220018

Name: Lantern for Mining Excavations

Group: Herramientas    U.M.: Unit (U)

Description: <No description>

Location: A01.52.D10

---

**Creation Details**

Created By: admin    On: 7/7/2010

---

[Turn Editing On](#)    [Delete Item](#)

To delete the item with name 'Lantern for Mining Excavations'

Index by Code file BEFORE item deletion

Item\_Code.txt - Notepad

File Edit Format View Help

013294	0
220018	1
709125	2
559207	3
409621	4
301298	5
660192	6

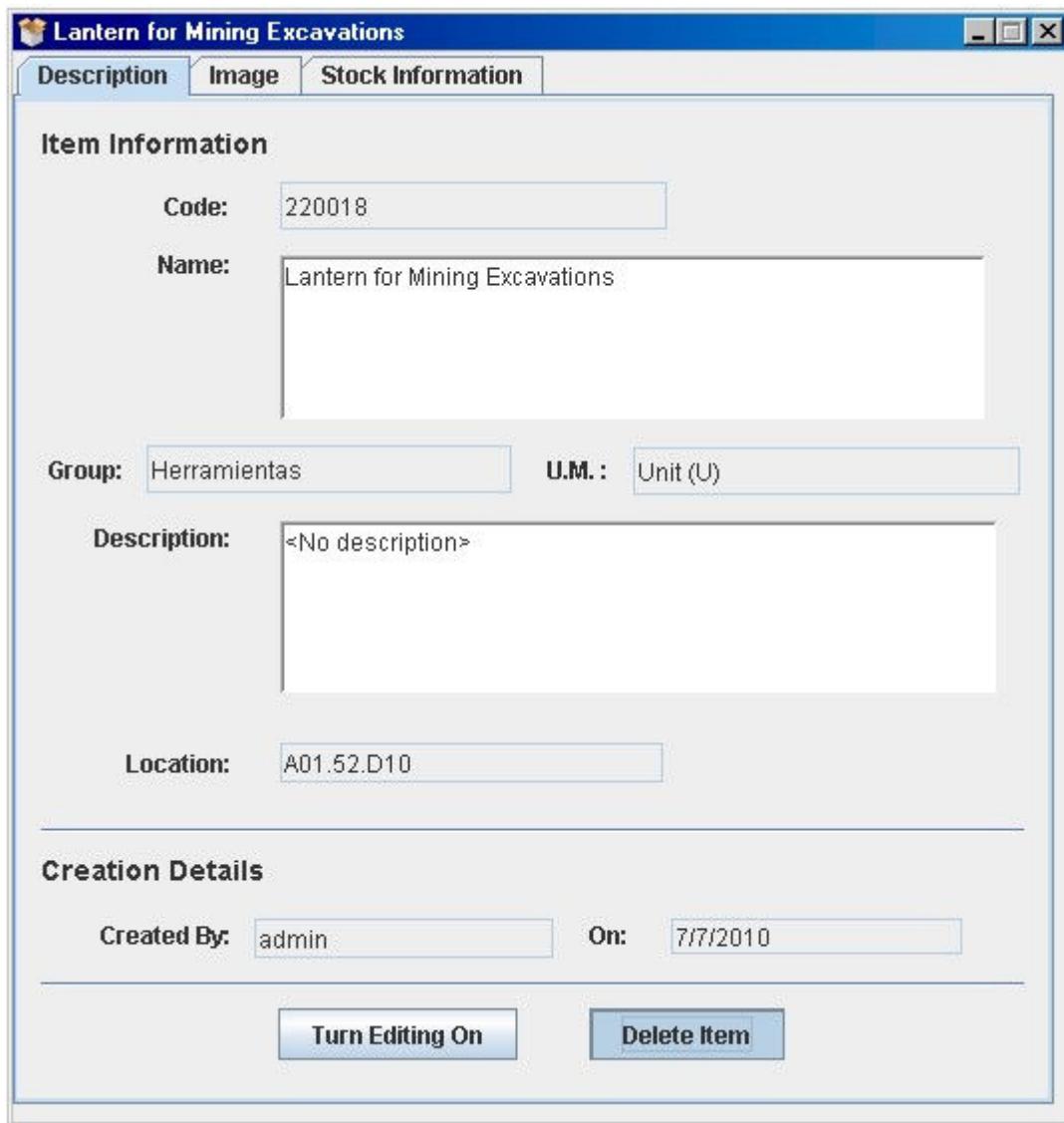
Index by Name file BEFORE item deletion

Item\_Name.txt - Notepad

File Edit Format View Help

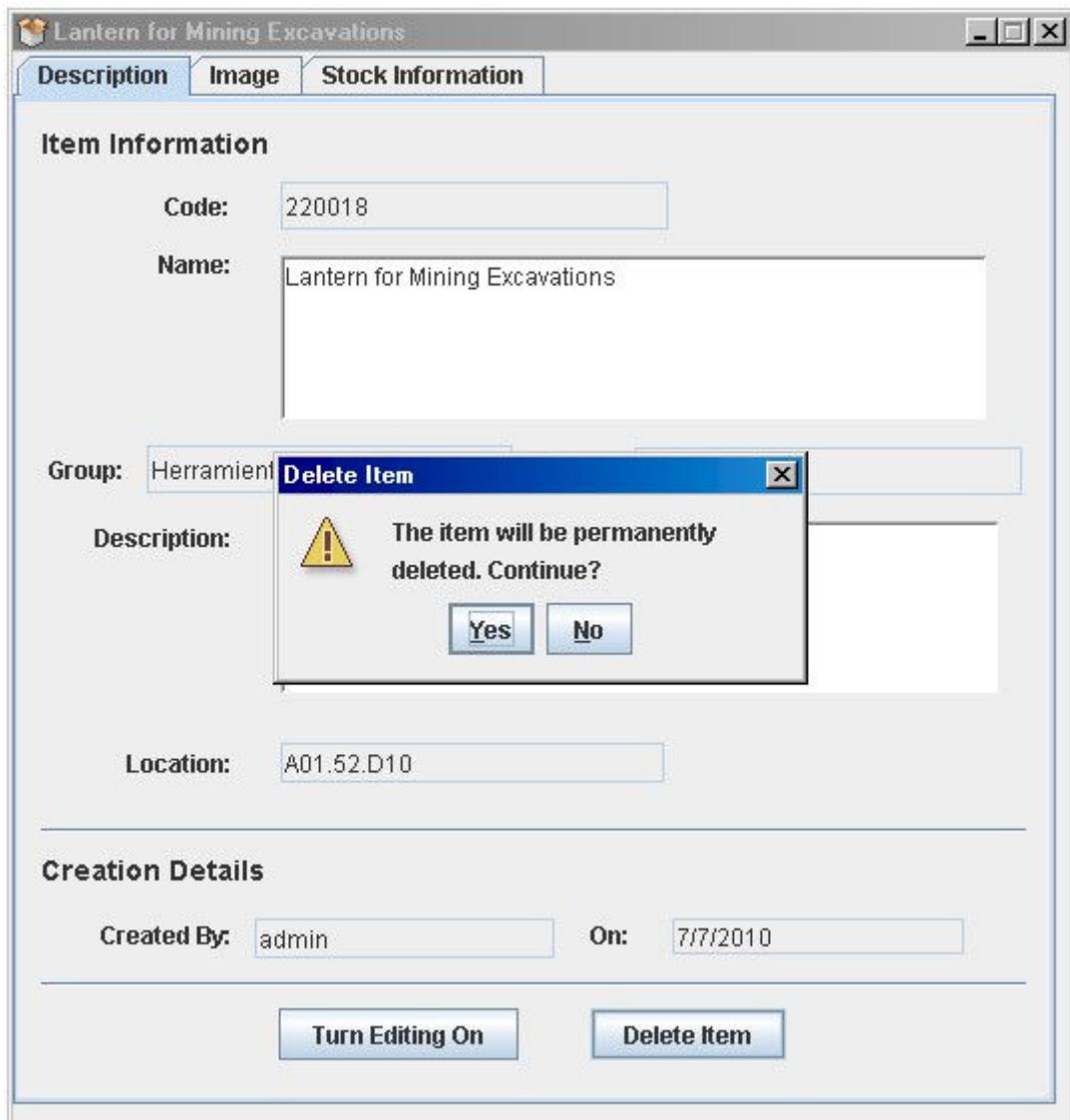
Laptop IBM ThinkPad R52 for Administrative Use	0
Lantern for Mining Excavations	1
Steel Nails in 1/16" width	2
1/3" Iron Bolts	3
Synthetic Glue for wood sheets	4
1/2" inch wide nylon rope	5
Left Handed wire Cutters	6

The 'Delete Item' button is pressed



*Result*

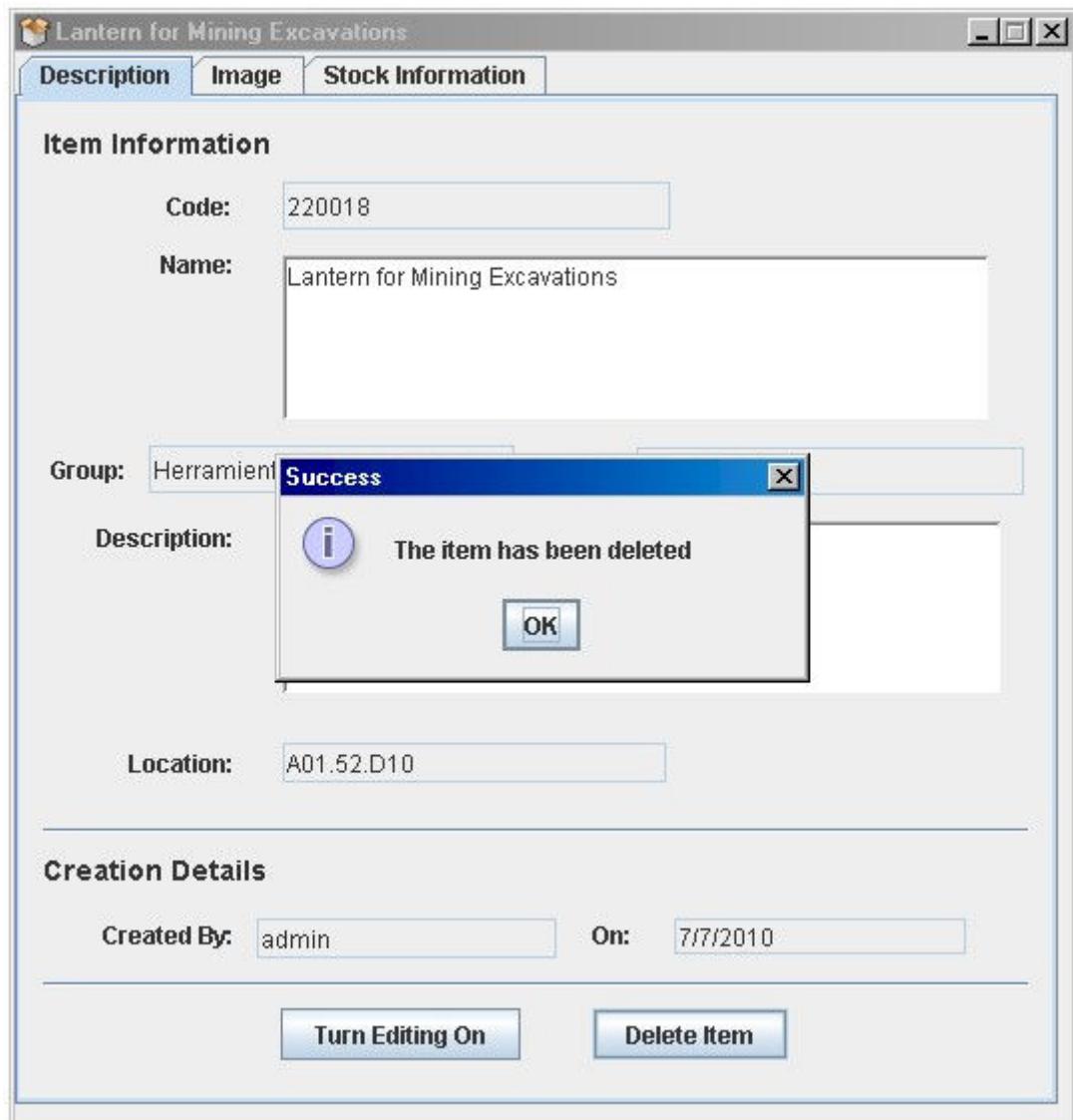
The user is prompted to delete the item



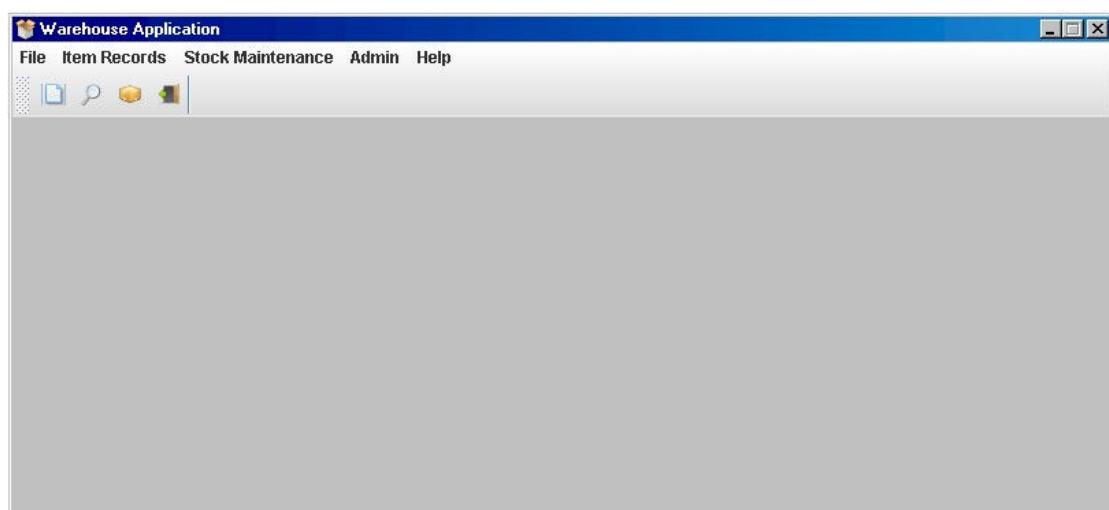
EITHER click 'Yes'

*Result*

User informed that the item was deleted



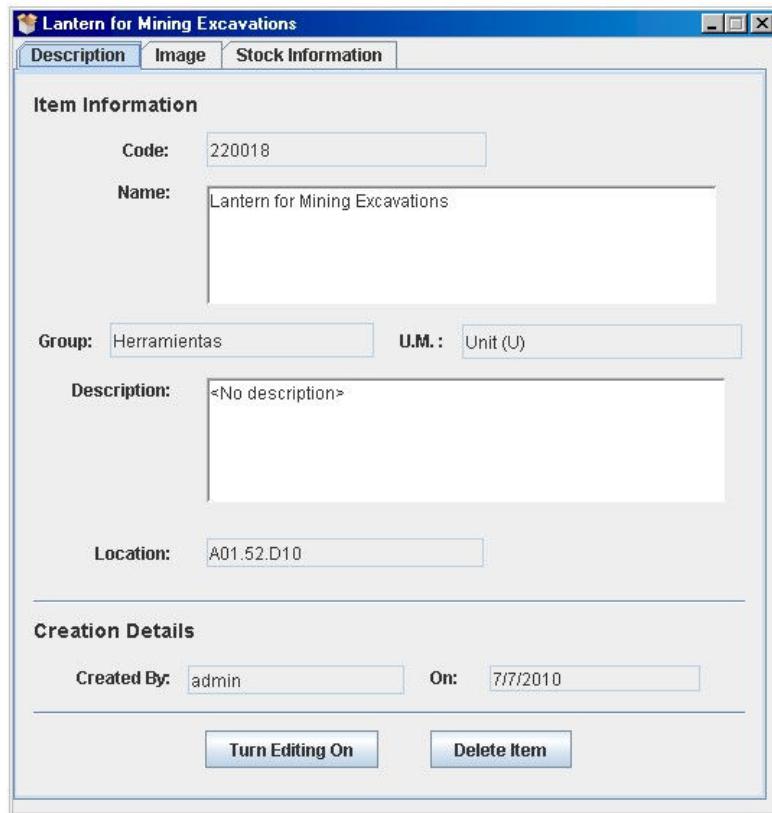
Item Description screen closes



OR click 'No'

*Result*

Item not deleted, Item Description screen still open



Item deleted from files

Index by Code file AFTER item deletion

The screenshot shows the 'Item\_Code.txt' Notepad file. The file contains the following code:

```
013294|0
709125|2
559207|3
409621|4
301298|5
660192|6
```

Index by Name file AFTER item deletion

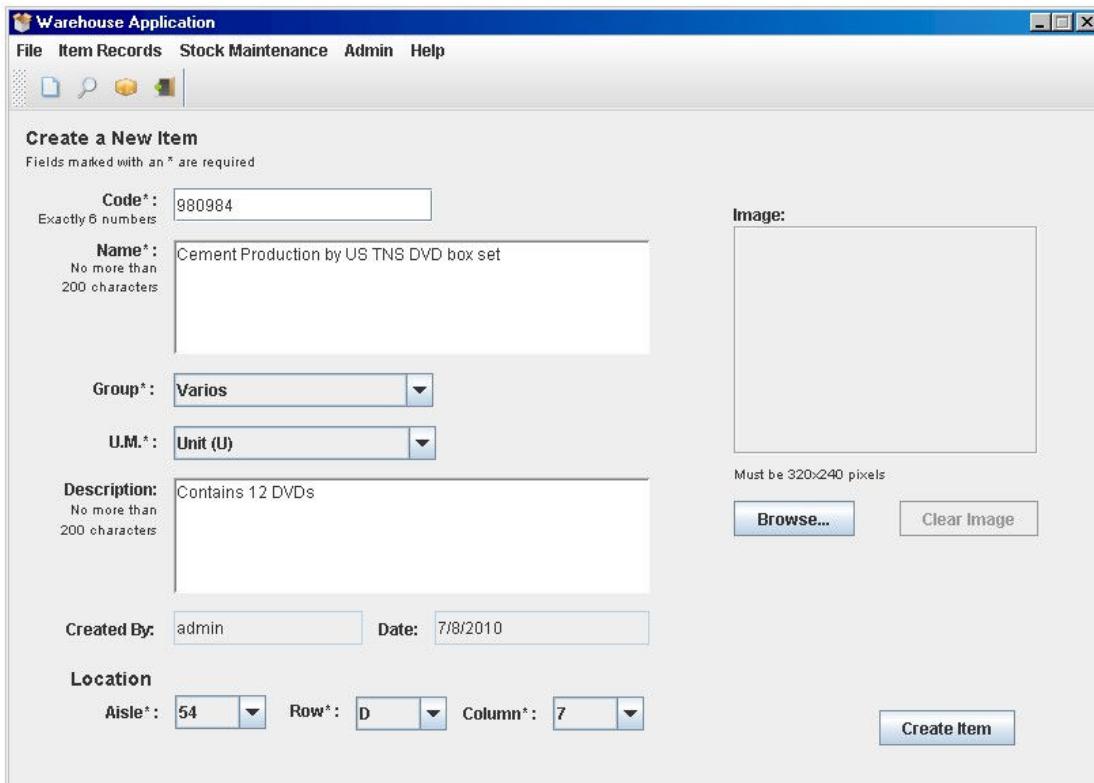
The screenshot shows the 'Item\_Name.txt' Notepad file. The file contains the following names:

```
Laptop IBM ThinkPad R52 for Administrative use|0
Steel Nails in 1/16" width|2
1/3" Iron Bolts|3
Synthetic Glue for wood sheets|4
1/2" inch wide nylon rope|5
Left Handed wire Cutters|6
```

## CREATING A NEW ITEM IN A FILE WITH SPACES

To avoid wasting space, a new item may be created in a record in the Items Random Access File that has been flagged as empty. A new item will be created in the index of the item that was just deleted above (1).

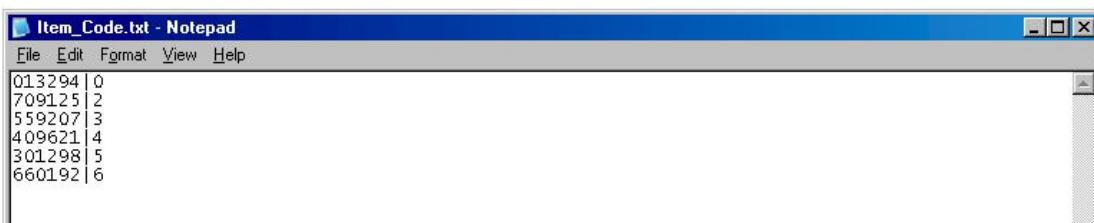
The item creation routine described above is carried out to create this item



The screenshot shows the 'Create a New Item' dialog box of the Warehouse Application. The window title is 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. The toolbar contains icons for file operations. The main form is titled 'Create a New Item' and contains the following fields:

- Code\***: 980984 (Exactly 6 numbers)
- Name\***: Cement Production by US TNS DVD box set (No more than 200 characters)
- Group\***: Varios
- U.M.\***: Unit (U)
- Description**: Contains 12 DVDs (No more than 200 characters)
- Created By**: admin
- Date**: 7/8/2010
- Location**: Aisle\*: 54, Row\*: D, Column\*: 7
- Image**: An empty placeholder area with instructions: 'Must be 320x240 pixels'. It includes 'Browse...' and 'Clear Image' buttons.
- Create Item** button at the bottom right.

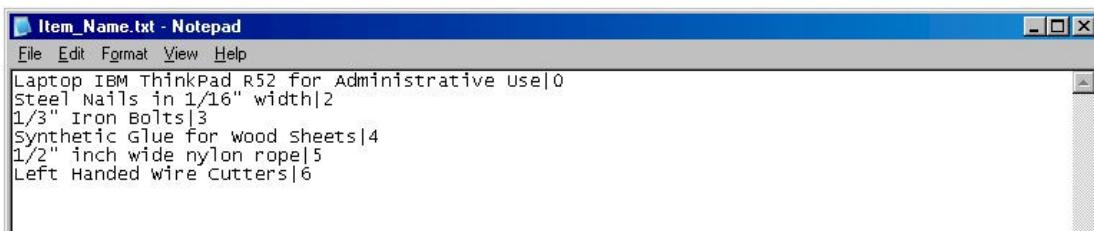
Index by Code file BEFORE item creation



The screenshot shows a Notepad window titled 'Item\_Code.txt - Notepad'. The menu bar includes 'File', 'Edit', 'Format', 'View', and 'Help'. The window displays the following index data:

Code	Index
013294	0
709125	2
559207	3
409621	4
301298	5
660192	6

Index by Name file BEFORE item creation



The screenshot shows a Notepad window titled 'Item\_Name.txt - Notepad'. The menu bar includes 'File', 'Edit', 'Format', 'View', and 'Help'. The window displays the following index data:

Name	Index
Laptop IBM ThinkPad R52 for Administrative Use	0
Steel Nails in 1/16" width	1
1/3" Iron Bolts	3
Synthetic Glue for wood sheets	4
1/2" inch wide nylon rope	5
Left Handed wire cutters	6

There is no item with ID 1. The new item will have that ID.

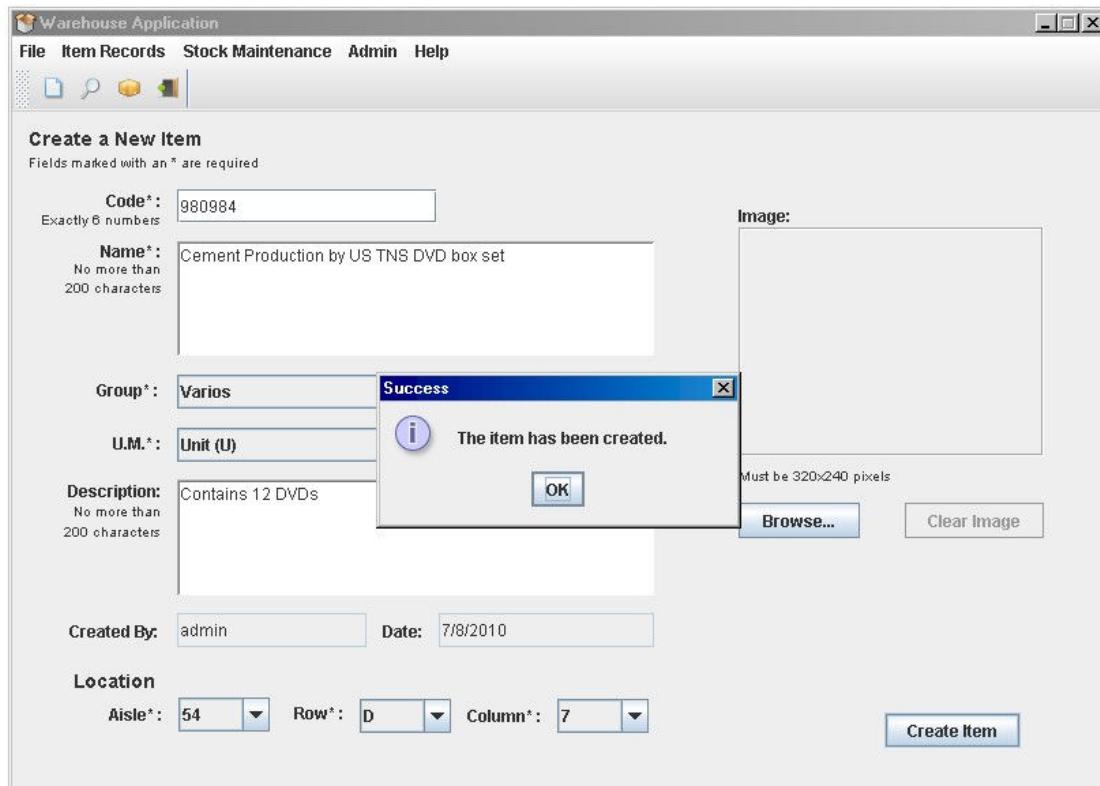
Click the 'Create Item' button

The screenshot shows the 'Warehouse Application' window with the title bar 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for file operations. The main area is titled 'Create a New Item' and contains the following fields:

- Code\***: 980984 (Exactly 6 numbers)
- Name\***: Cement Production by US TNS DVD box set (No more than 200 characters)
- Image**: A placeholder area with instructions: 'Must be 320x240 pixels' and buttons for 'Browse...' and 'Clear Image'.
- Group\***: Varios
- U.M.\***: Unit (U)
- Description**: Contains 12 DVDs (No more than 200 characters)
- Created By**: admin
- Date**: 7/8/2010
- Location**: Aisle\*: 54, Row\*: D, Column\*: 7
- Create Item** button (highlighted in blue)

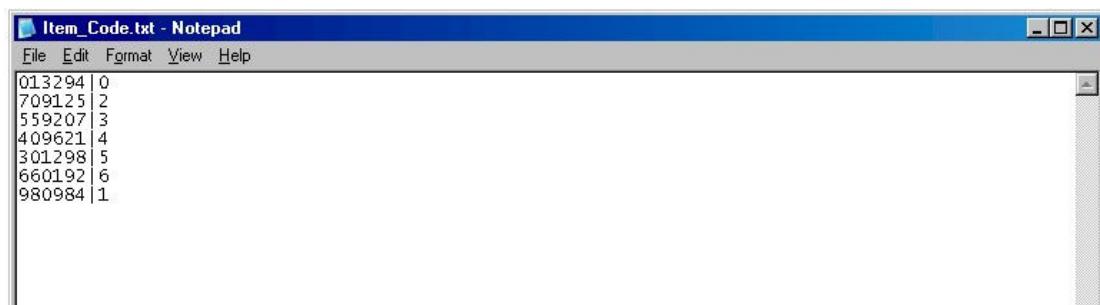
### Result

User informed that the item was created



Item created in index files with correct ID

Index by Code file AFTER item creation



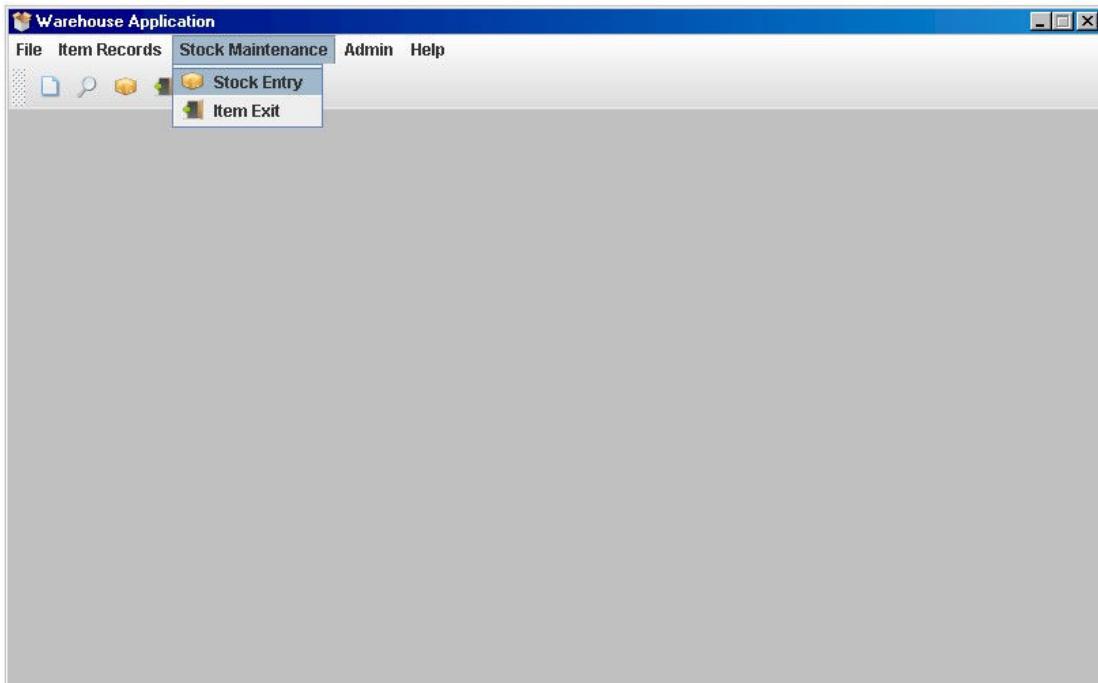
Index by Name file AFTER item creation



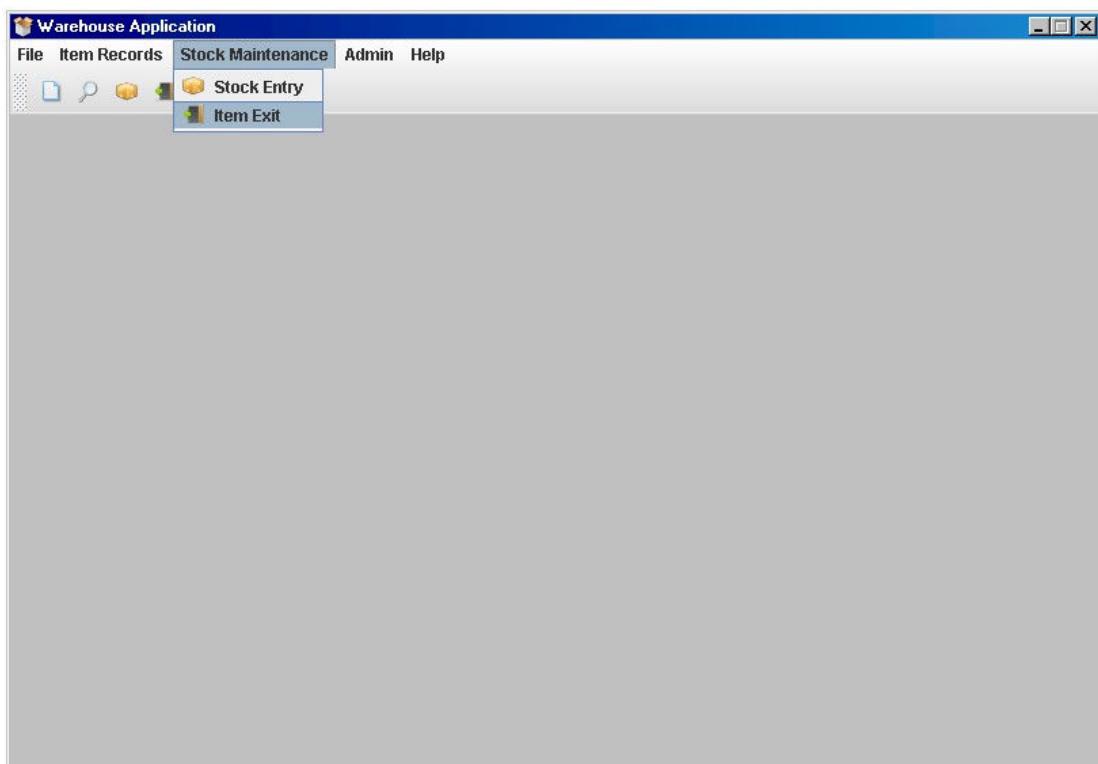
## LOADING THE MULTIPLE ITEM SEARCH SCREEN

EITHER

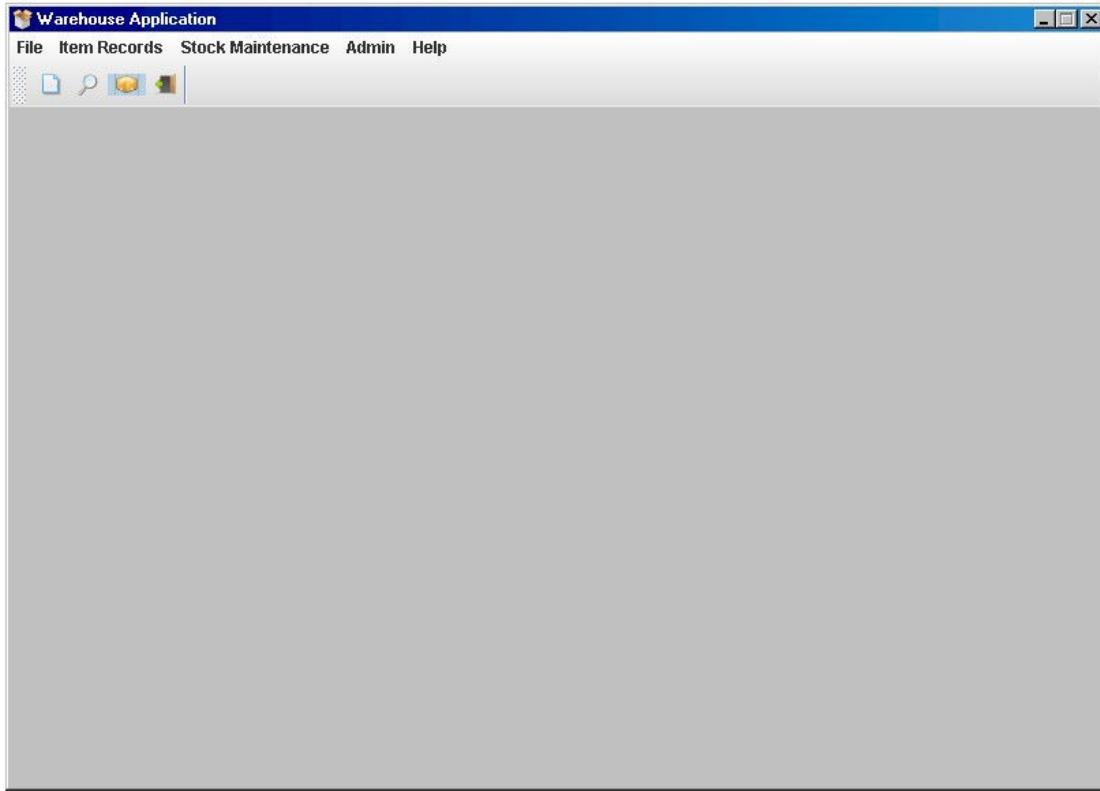
Clicking Stock Maintenance > Item Entry from the Menu Bar



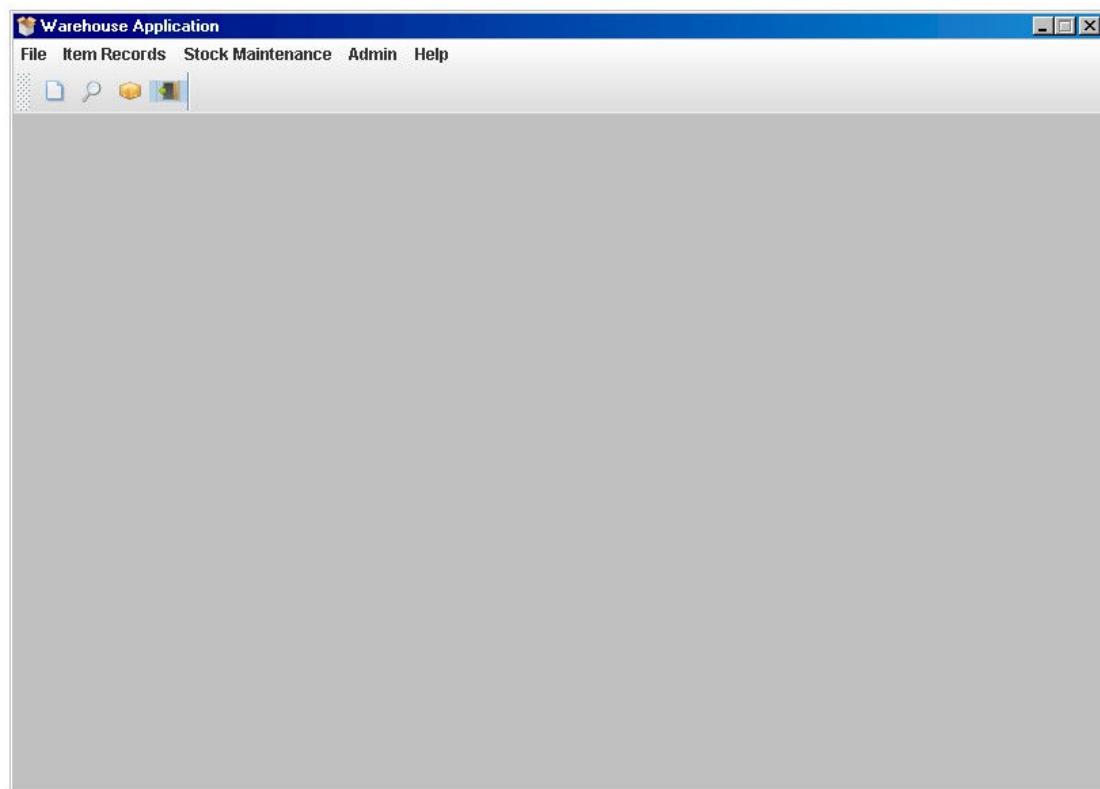
OR Clicking Stock Maintenance > Item Exit from the Menu Bar



OR Clicking the Item Entry button from the toolbar



OR Clicking the Item Exit button from the toolbar



*Result*

The multiple item search screen loads

The screenshot shows the 'Warehouse Application' window with the title bar 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. Below the menu is a toolbar with icons for file operations. The main interface is divided into two sections:

- 1. Multiple Item Search:** A panel on the left containing:
  - A dropdown menu labeled 'Search by:' with 'Code' selected.
  - An input field labeled 'Code:' with placeholder text 'Only digits'.
  - A 'Search' button.
  - A note: 'Double click on an item to view more information'.
  - A table with columns: Code, Name, U.M., In Stock. The table body is empty.
  - A horizontal scrollbar at the bottom of the table area.
  - A 'Add' button at the bottom right.
- 2. Item List:** A panel on the right containing:
  - A note: 'To remove an item, select it and click "remove"'.
  - A large empty rectangular area representing the item list.
  - A 'Remove' button located below the list area.
  - A 'Next Step' button at the bottom right.

## SEARCHING FOR ONE OR MORE ITEMS BY CODE

Select Code from the drop-down list and the code search boxes load

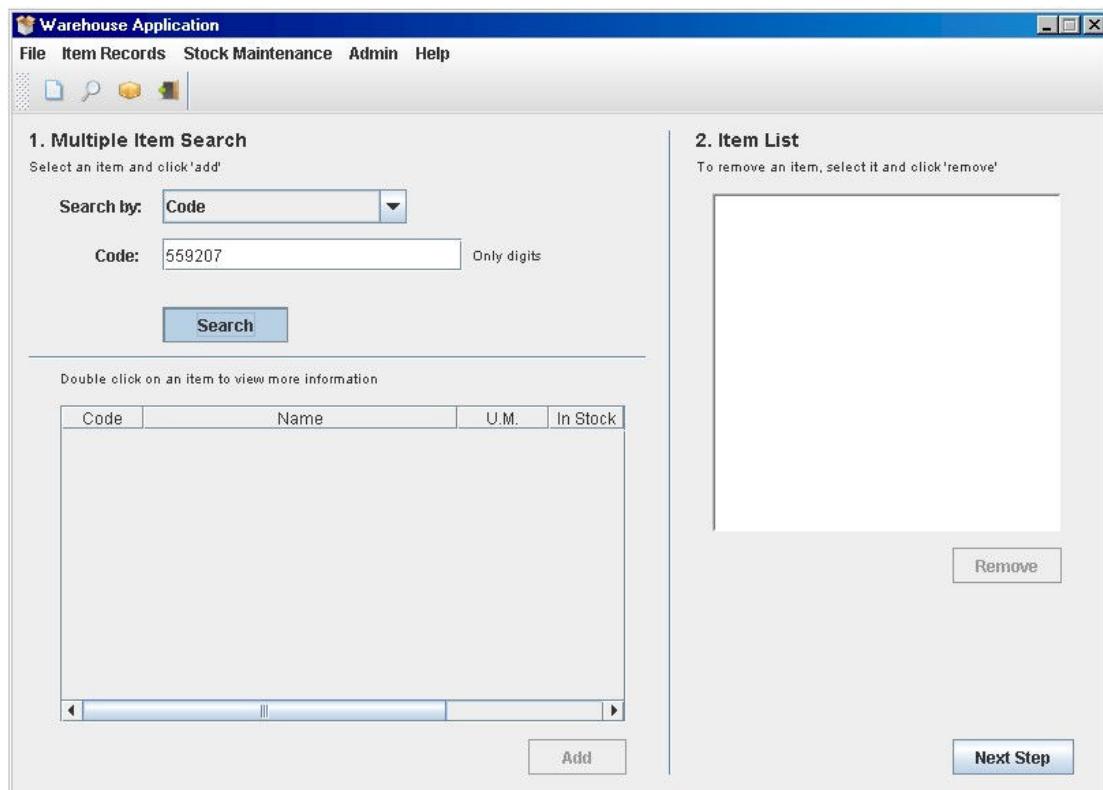
### CORRECT INPUT

1. Searching for an item that exists

*Test*

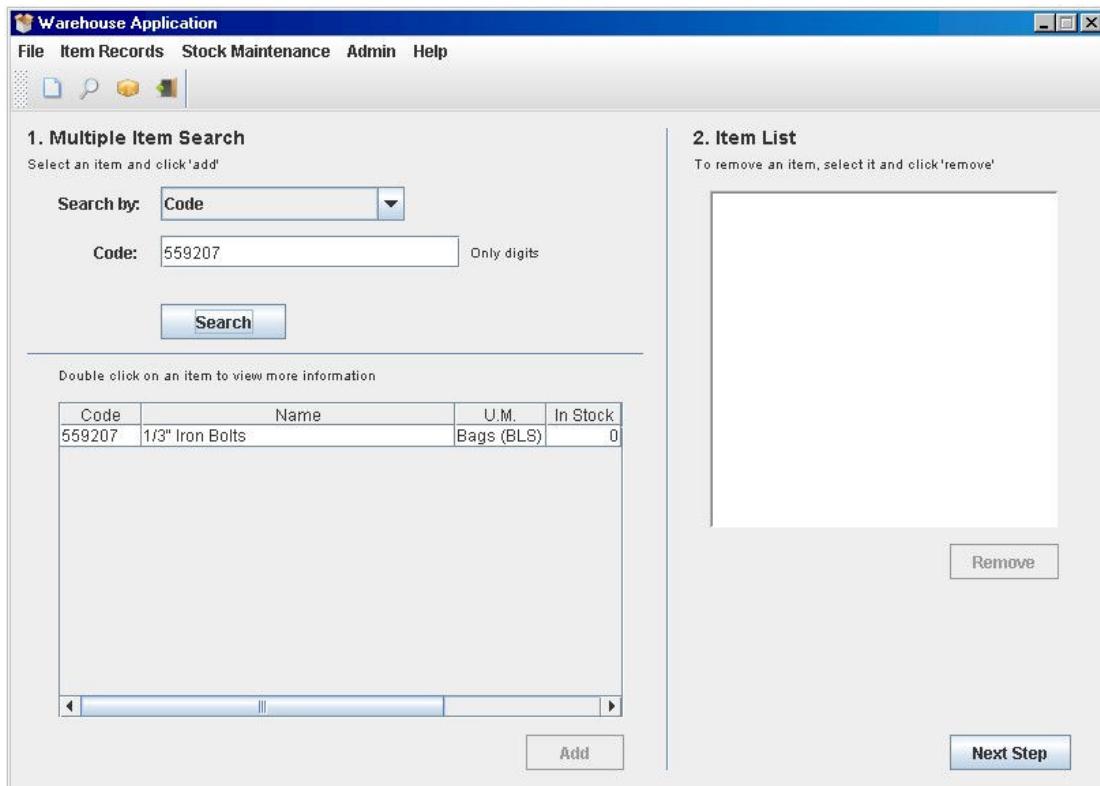
Search by: Code  
Code: 559207 [the search query]

The 'Search' button is clicked



*Results*

Appropriate result is displayed.

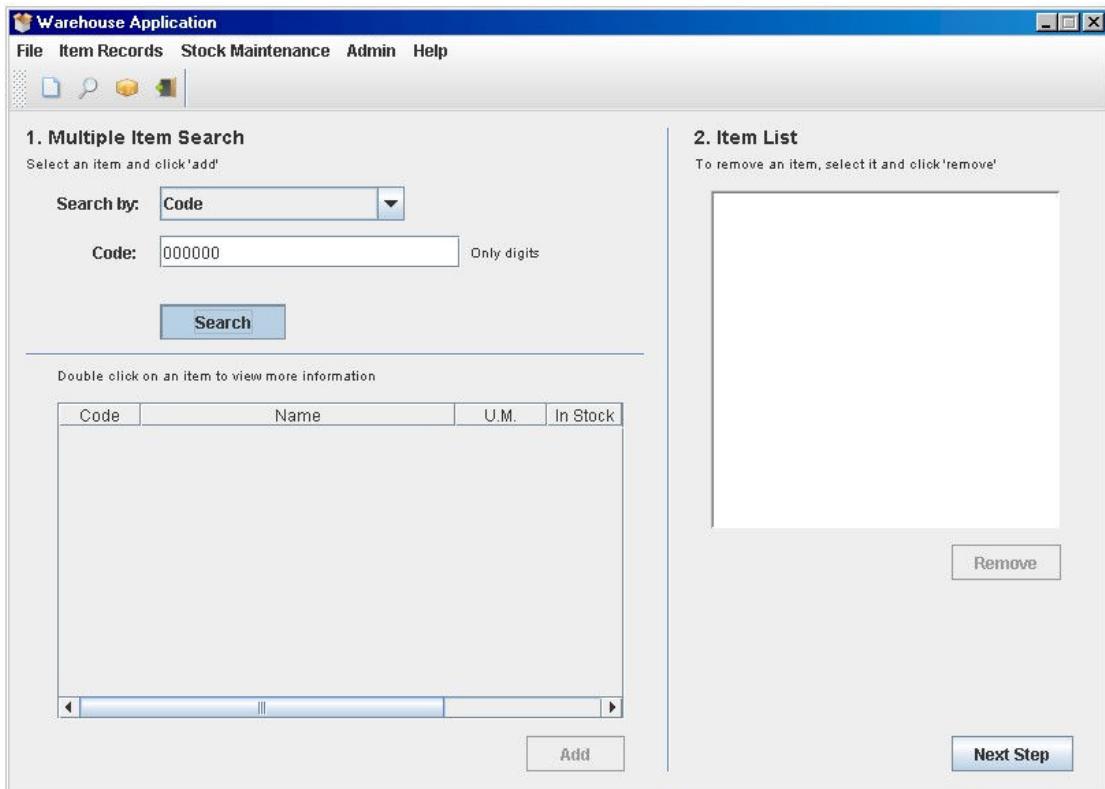


## 2. Searching for an item that does not exist

*Test*

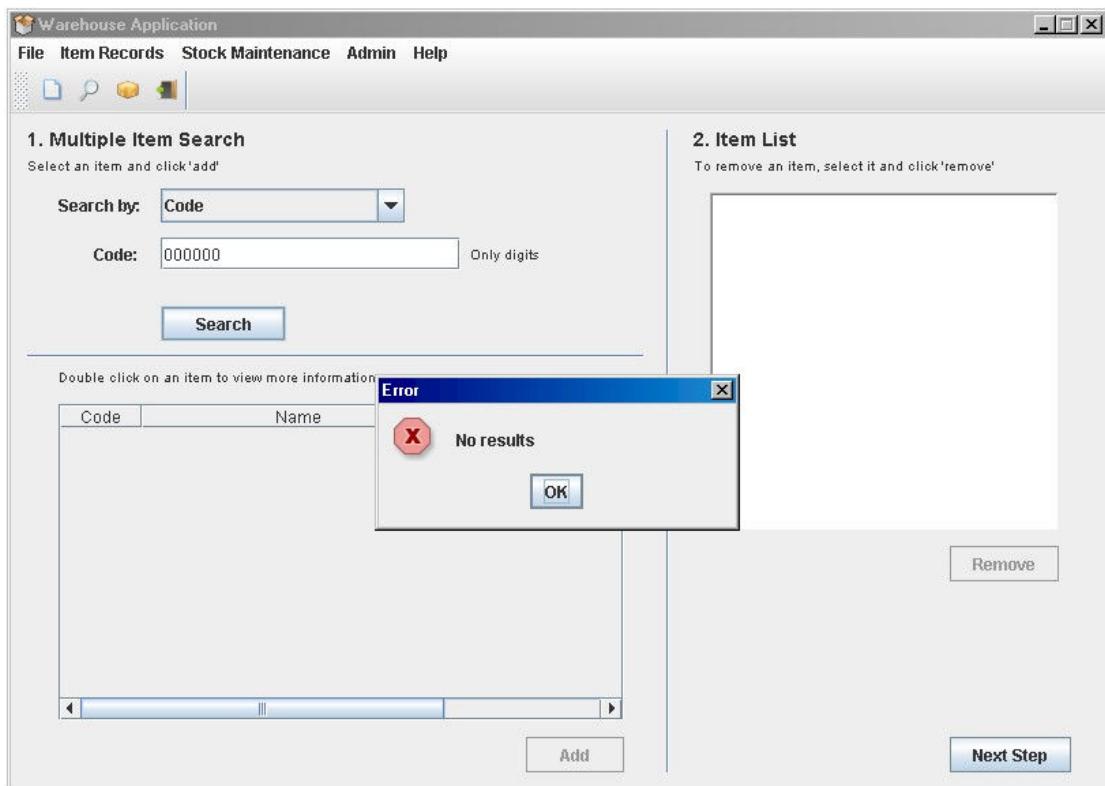
Search by: Code  
Code: 000000 [the search query]

The 'Search' button is clicked



### Result

No results message displayed



## **INCORRECT INPUT**

1. A query containing non-digits

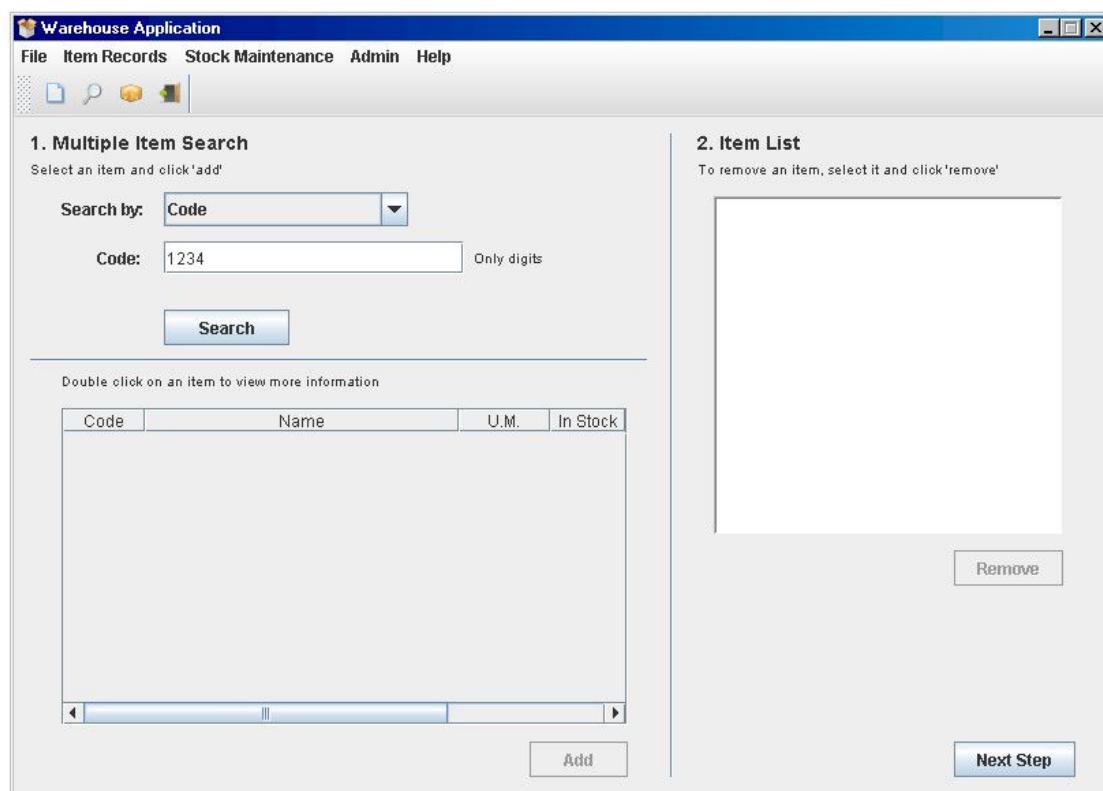
Test:

Search by: Code  
Name: ab1234

The 'Search' button is pressed

*Result*

Non-digit characters are not entered



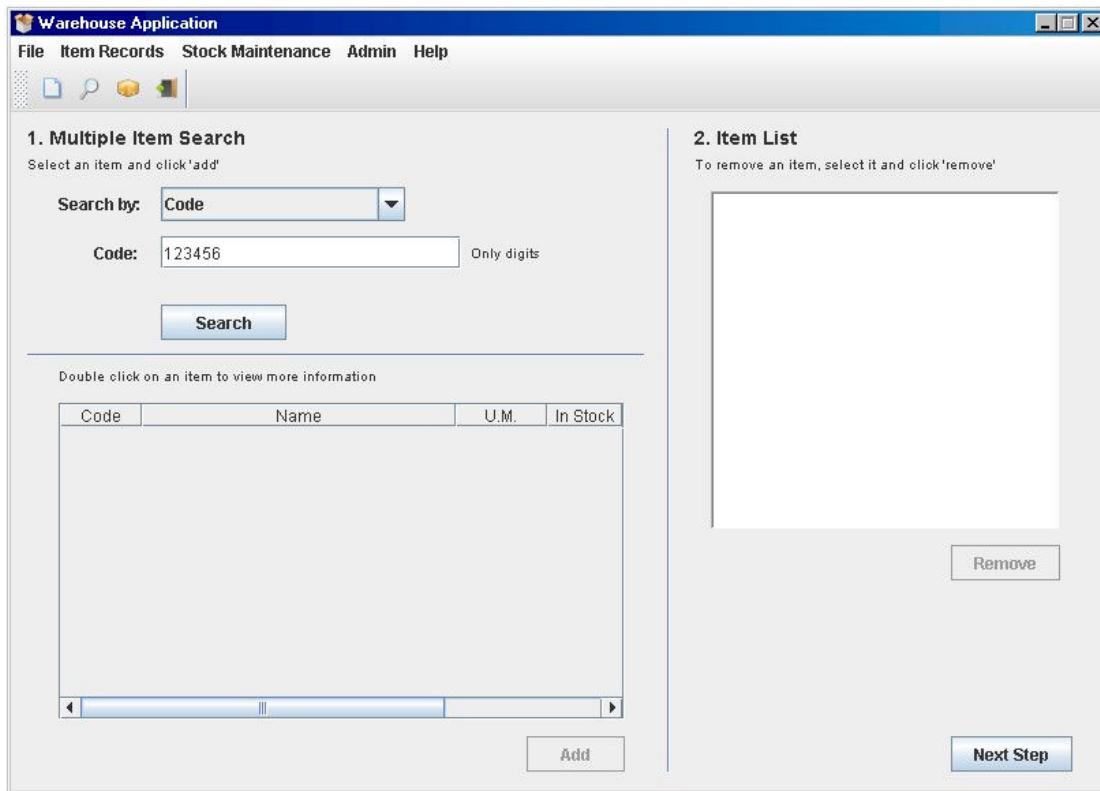
2. A query longer than 6 digits

Test:

Search by: Code  
Name: 12345678 [8 digits]

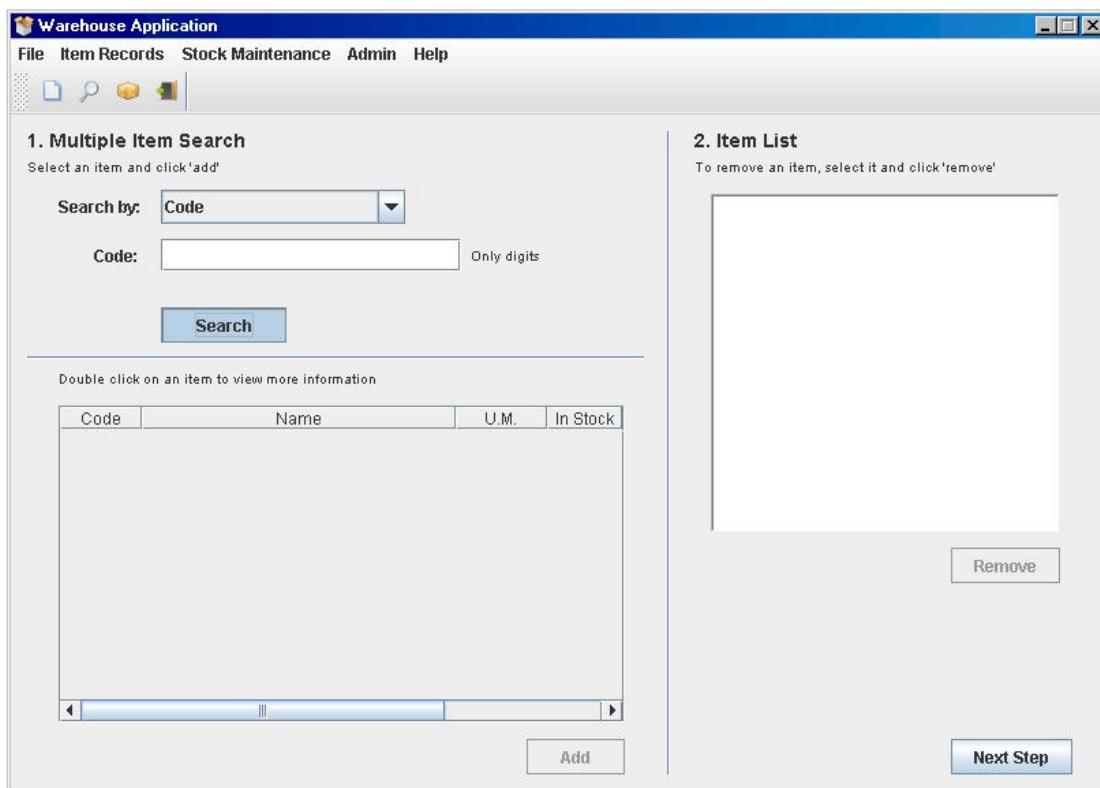
*Result*

Query truncated at 6 digits



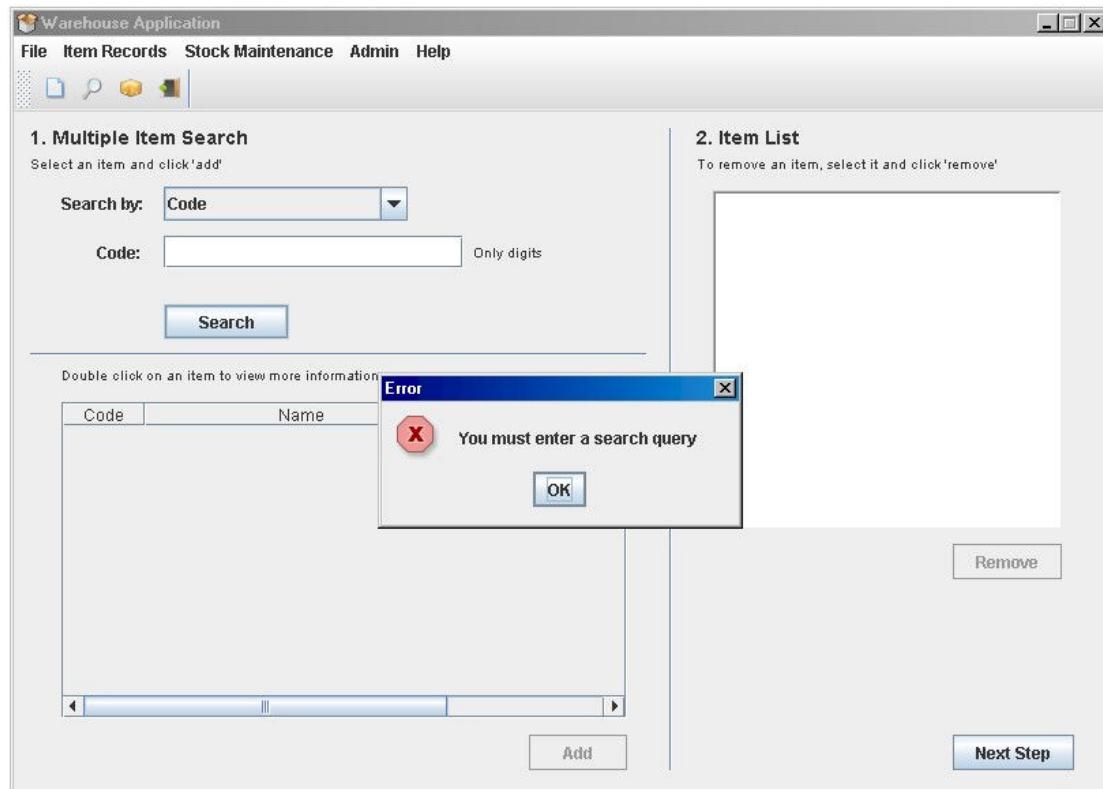
### 3. An empty query

The 'Search' button is pressed



## Result

User asked to enter a search query

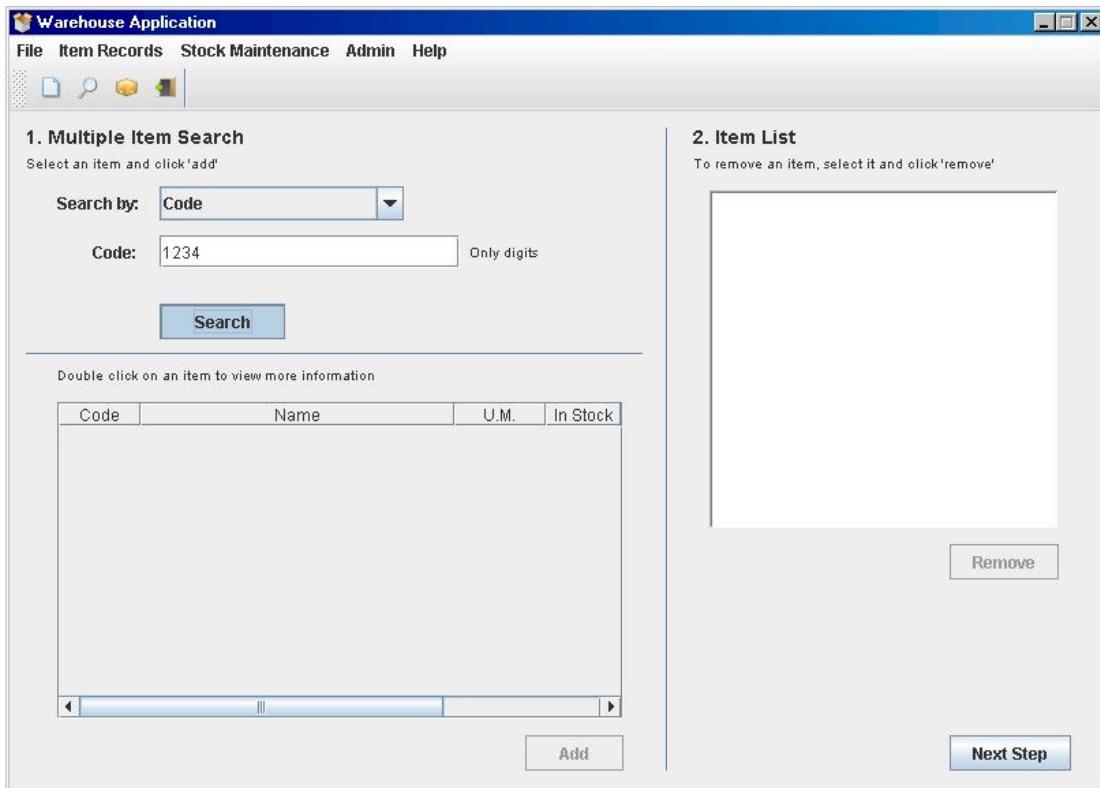


## 4. A query under 6 digits

Test:

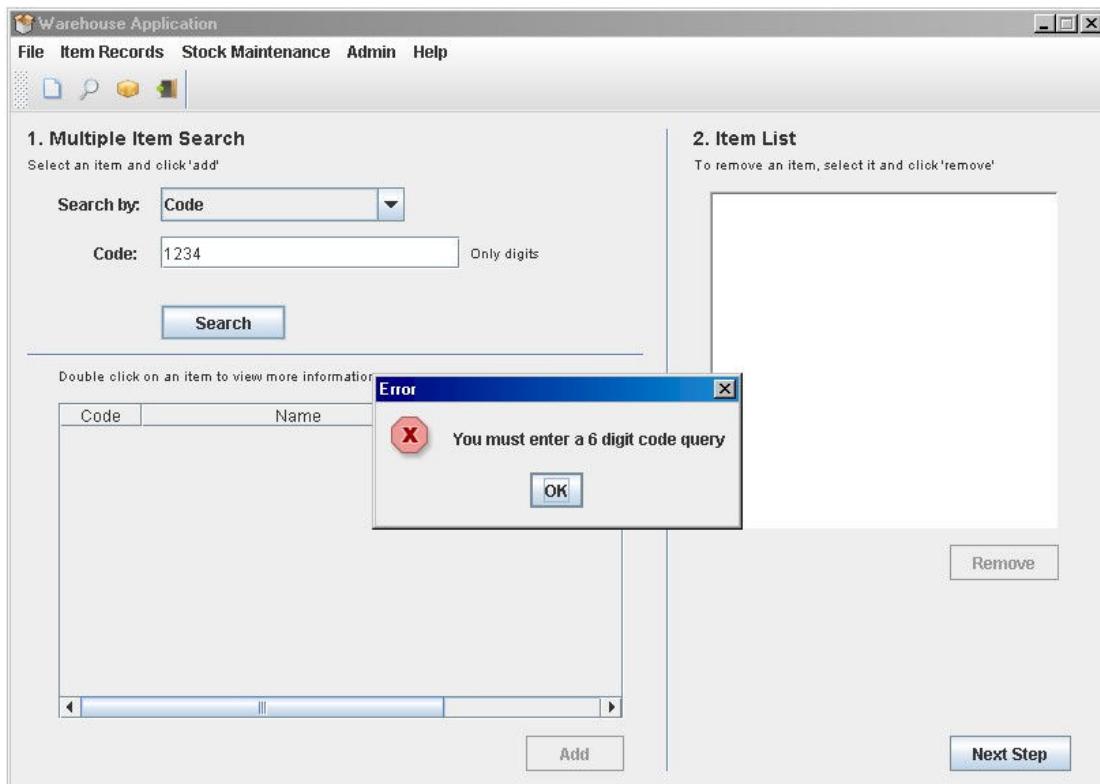
Search by: Code  
Name: 1234 [4 digits]

The 'Search' button is pressed



### Result

User informed that input is invalid



## SEARCHING FOR ONE OR MORE ITEMS BY PARTIAL NAME

Select Name from the drop-down list and the name search boxes load, partial match radio button selected

### CORRECT INPUT

1. Searching for an item that exists

*Test*

Search by: Name  
Name: s [the search query]  
Type: Partial Match

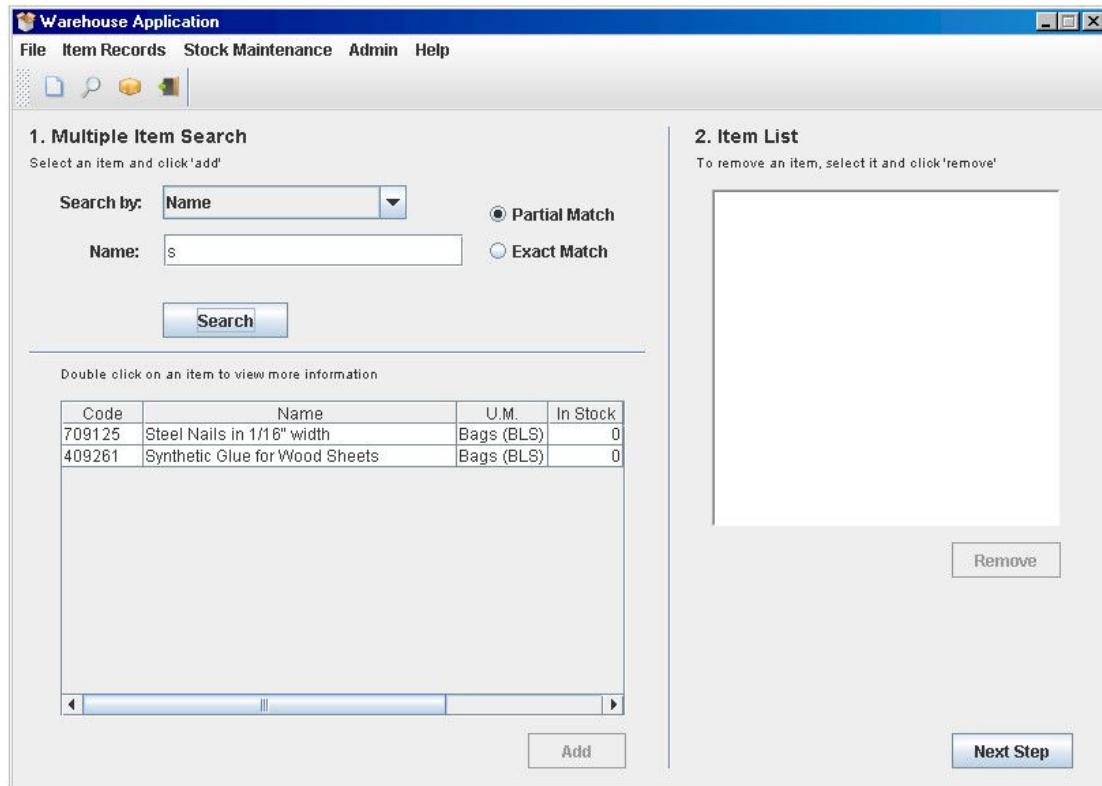
The 'Search' button is pressed

The screenshot shows the 'Warehouse Application' window. The menu bar includes File, Item Records, Stock Maintenance, Admin, and Help. The toolbar has icons for New, Open, Save, Print, and Exit. The main window is divided into two sections:

- 1. Multiple Item Search:** A form with "Search by: Name" dropdown, "Name: s" input field, and radio buttons for "Partial Match" (selected) and "Exact Match". A "Search" button is below. A note says "Select an item and click 'add'". Below the form is a table header: "Code", "Name", "U.M.", "In Stock". A scrollable list area is below the header.
- 2. Item List:** A note says "To remove an item, select it and click 'remove'". A large empty rectangular area for displaying results, with a "Remove" button at the bottom right. A "Next Step" button is at the bottom right of this section.

*Result*

Appropriate results display

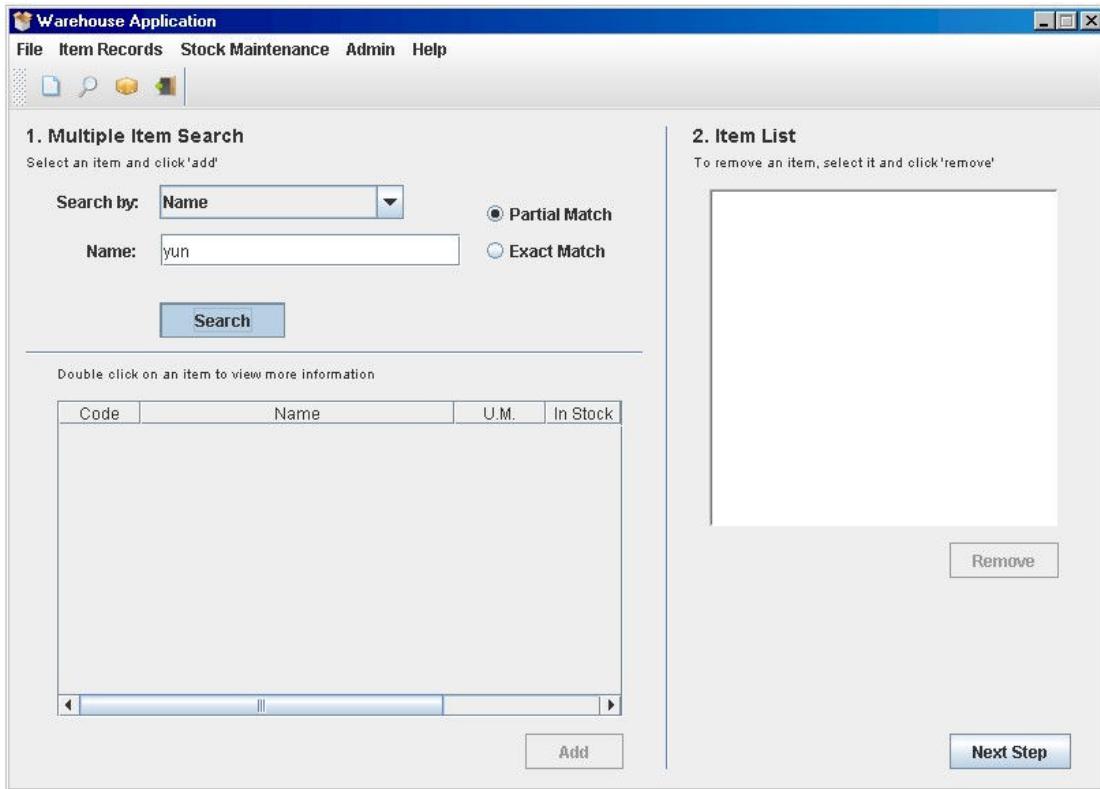


## 2. Searching for an item that does not exist

*Test*

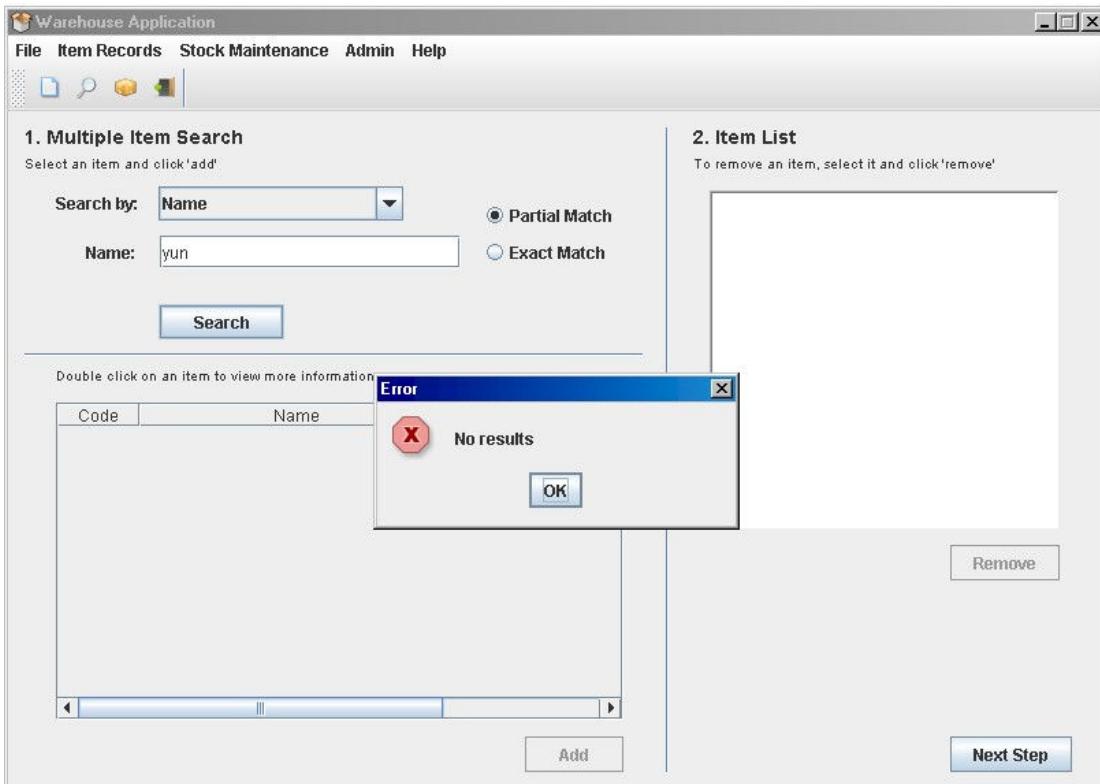
Search by: Name  
Name: yun [the search query]  
Type: Partial Match

The 'Search' button is pressed



## Results

No results dialog displayed



## INCORRECT INPUT

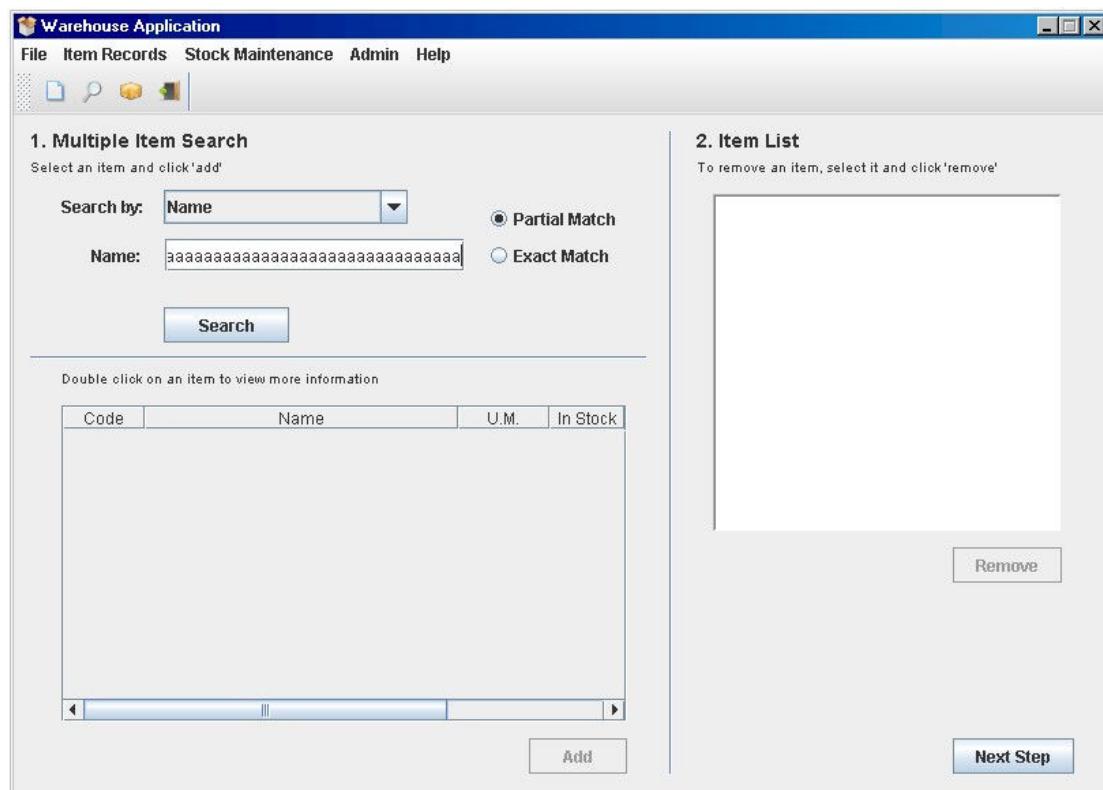
1. A query more than 200 characters in length

Test:

Search by: Name  
Name:  
aa  
aaa  
aa  
aa  
[210 characters]  
Type: Partial Match

*Result*

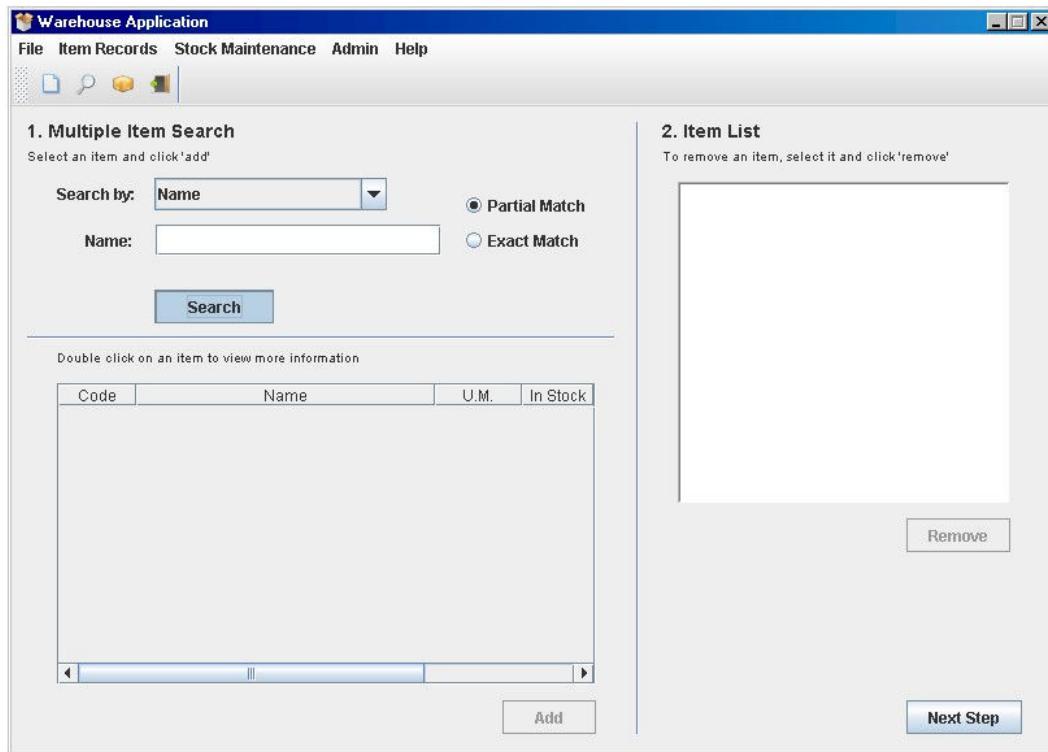
The query truncates at 200 characters



2. No (i.e. empty) search query

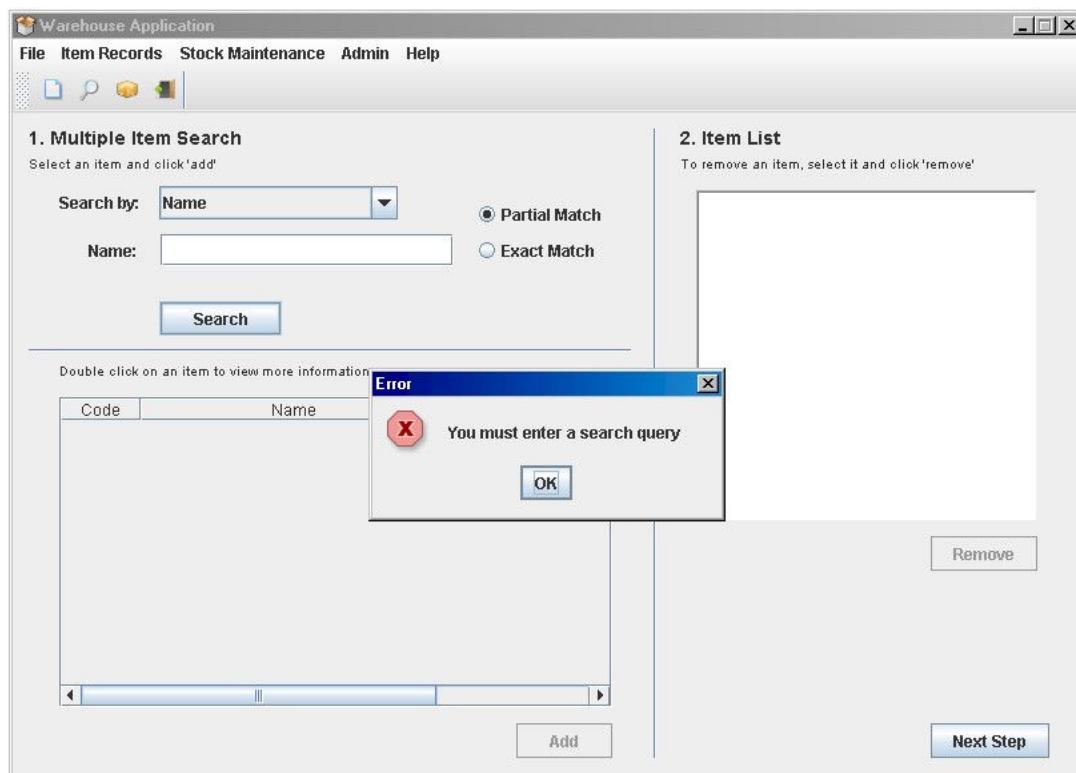
Search by: Name  
Name: [empty]  
Type: Partial Match

The 'Search' button is pressed



*Result*

User asked to enter a search query



## SEARCHING FOR ONE OR MORE ITEMS BY EXACT NAME

Select Name from the drop-down list and the name search boxes load, exact match radio button selected

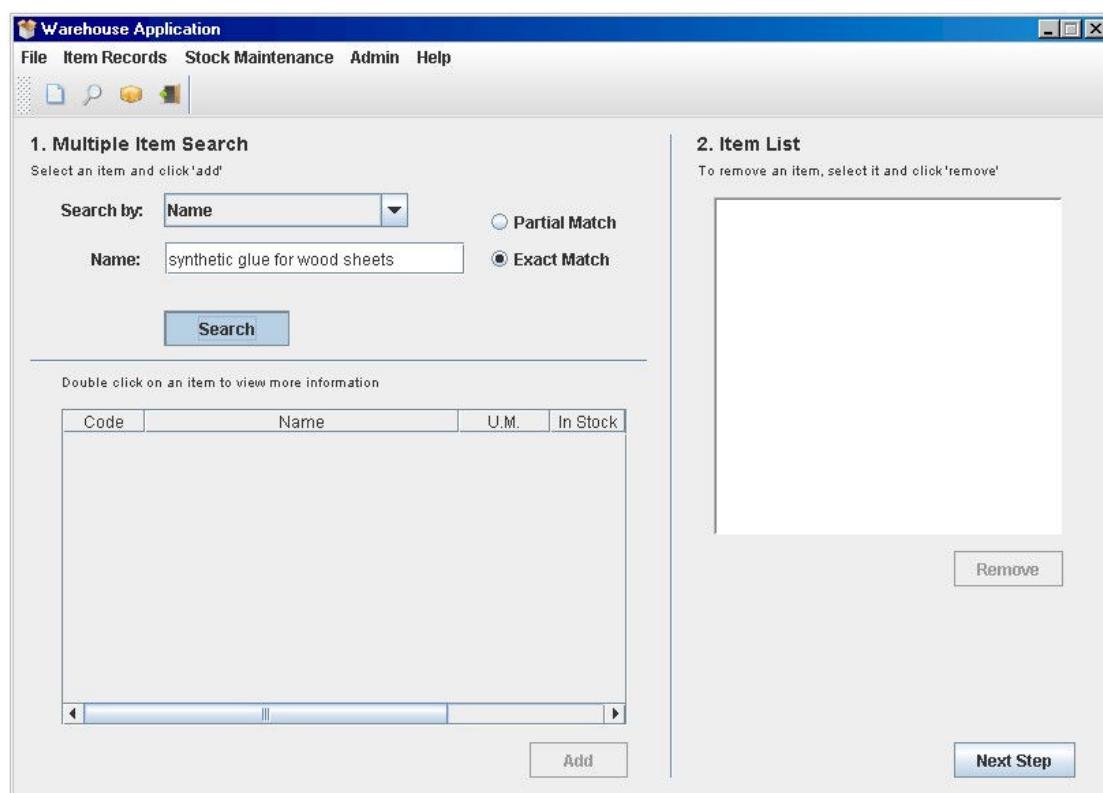
### CORRECT INPUT

1. Searching for an item that exists

*Test*

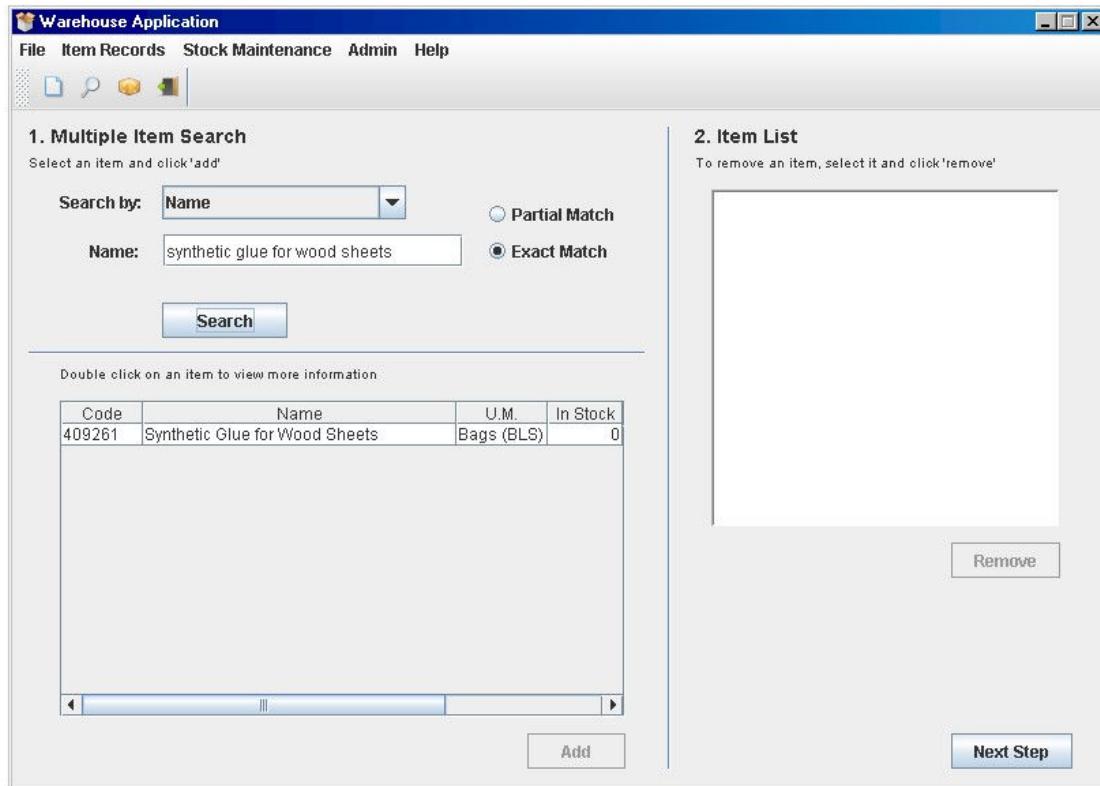
Search by: Name  
Name: synthetic glue for wood sheets [the search query]  
Type: Exact Match

The 'Search' button is pressed



*Result*

The appropriate search result is displayed

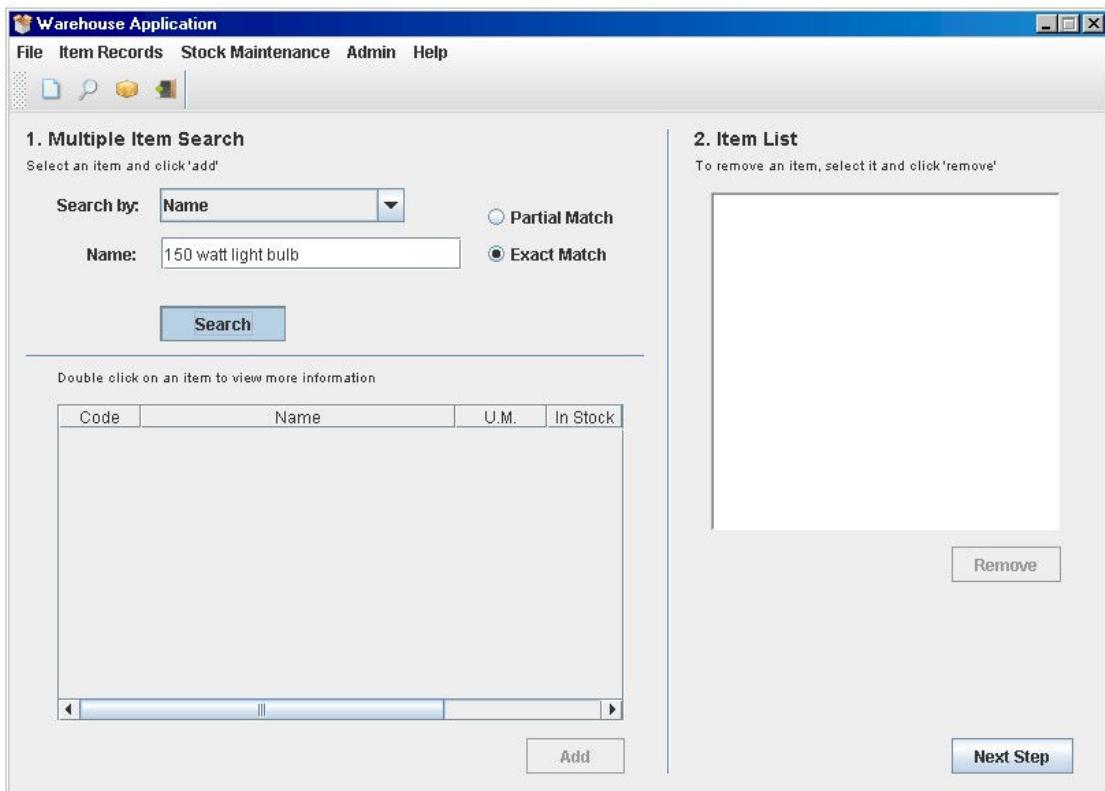


## 2. Searching for an item that does not exist

*Test*

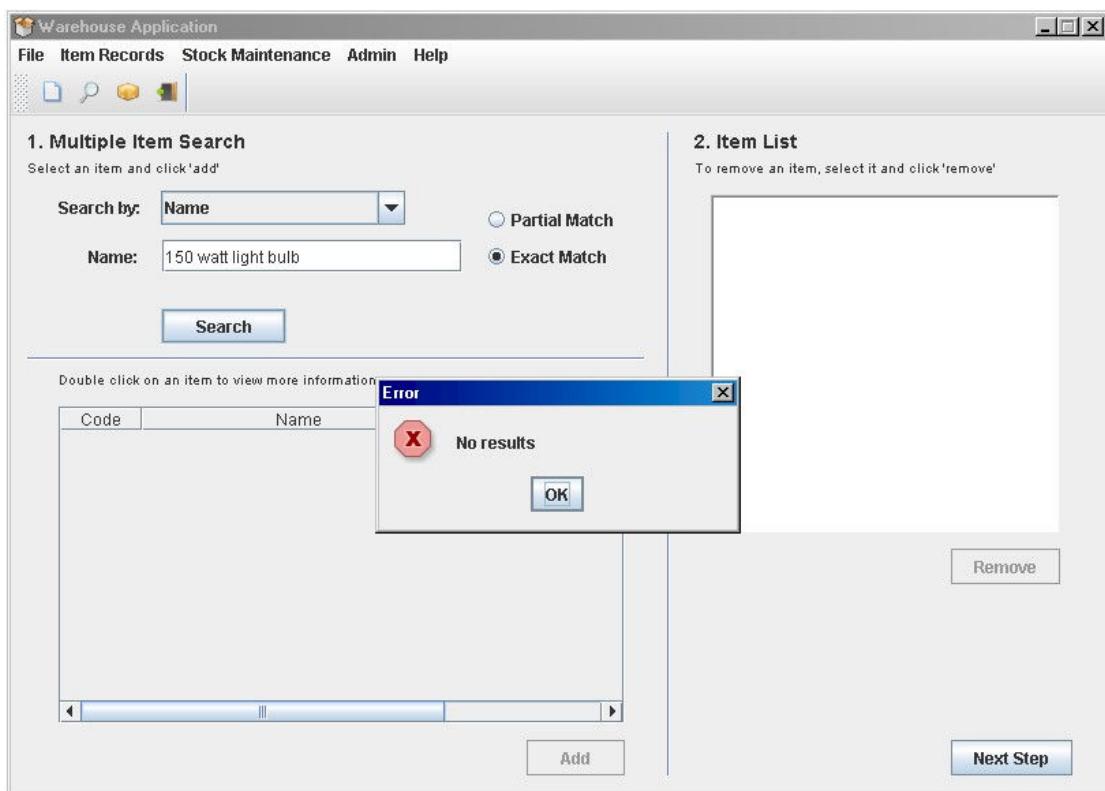
Search by: Name  
Name: 150 watt light bulb [the search query]  
Type: Exact Match

The 'Search' button is pressed



### Result

No results message displayed



## INCORRECT INPUT

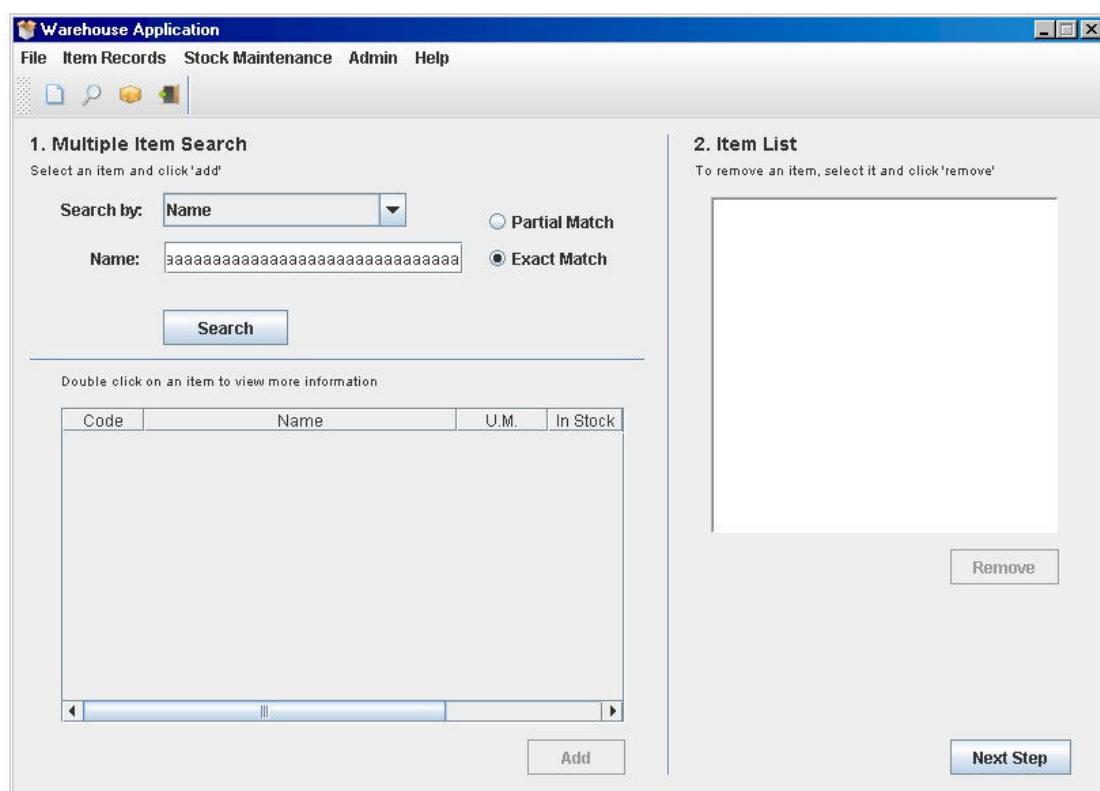
1. A query more than 200 characters in length

Test:

Search by: Name  
Name:  
aaa  
aaa  
aaa  
aaa  
[210 characters]  
Type: Exact Match

*Result*

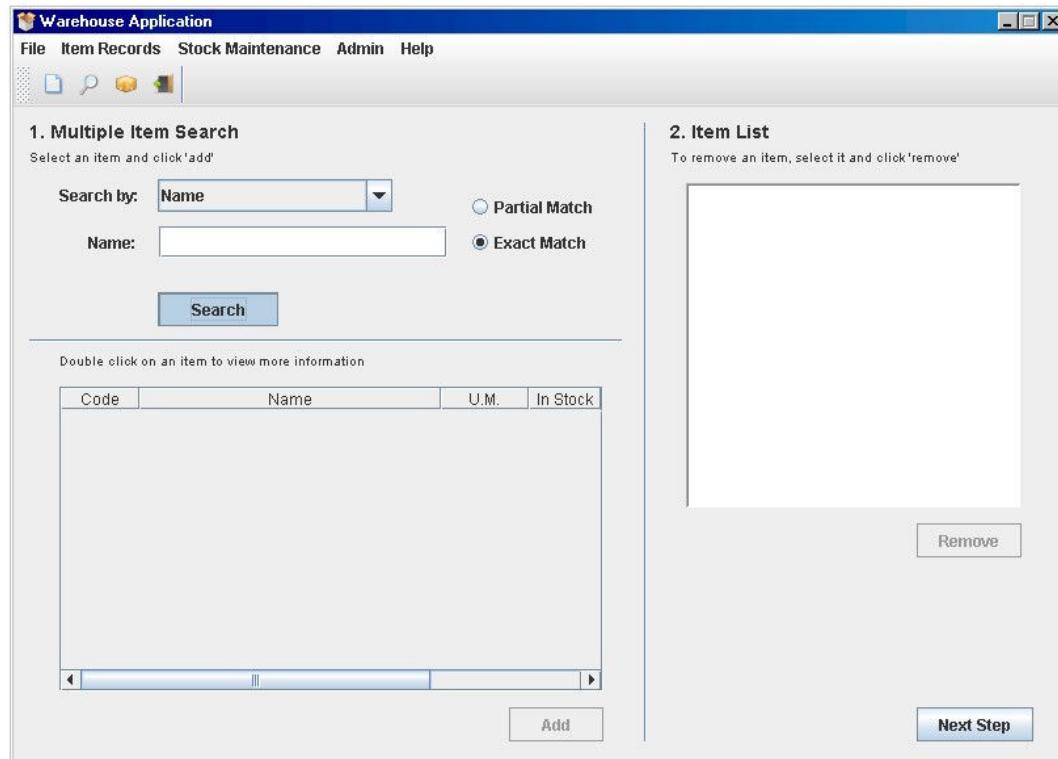
The query truncates at 200 characters



2. No (i.e. empty) search query

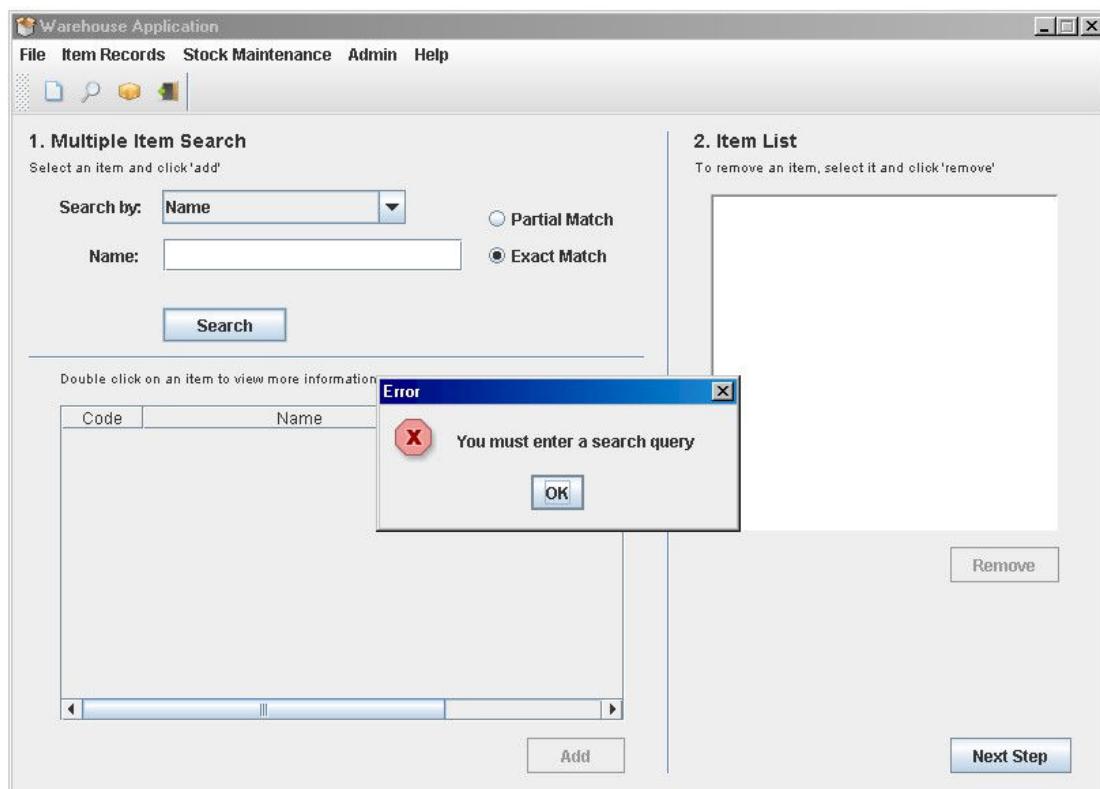
Search by: Name  
Name: [empty]  
Type: Exact Match

The 'Search' button is pressed



### Result

User asked to enter a search query



## SEARCHING FOR ONE OR MORE ITEMS BY GROUP

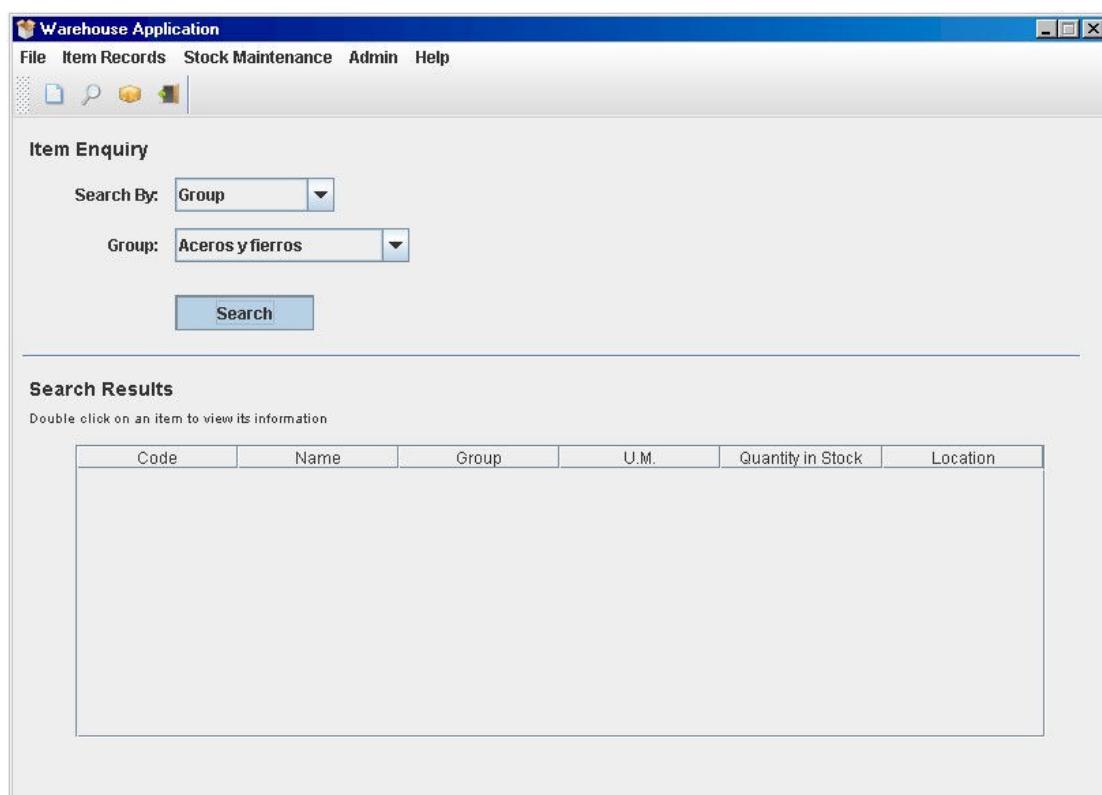
Select Group from the drop-down list and the group search boxes load

1. Searching for an item that exists

*Test*

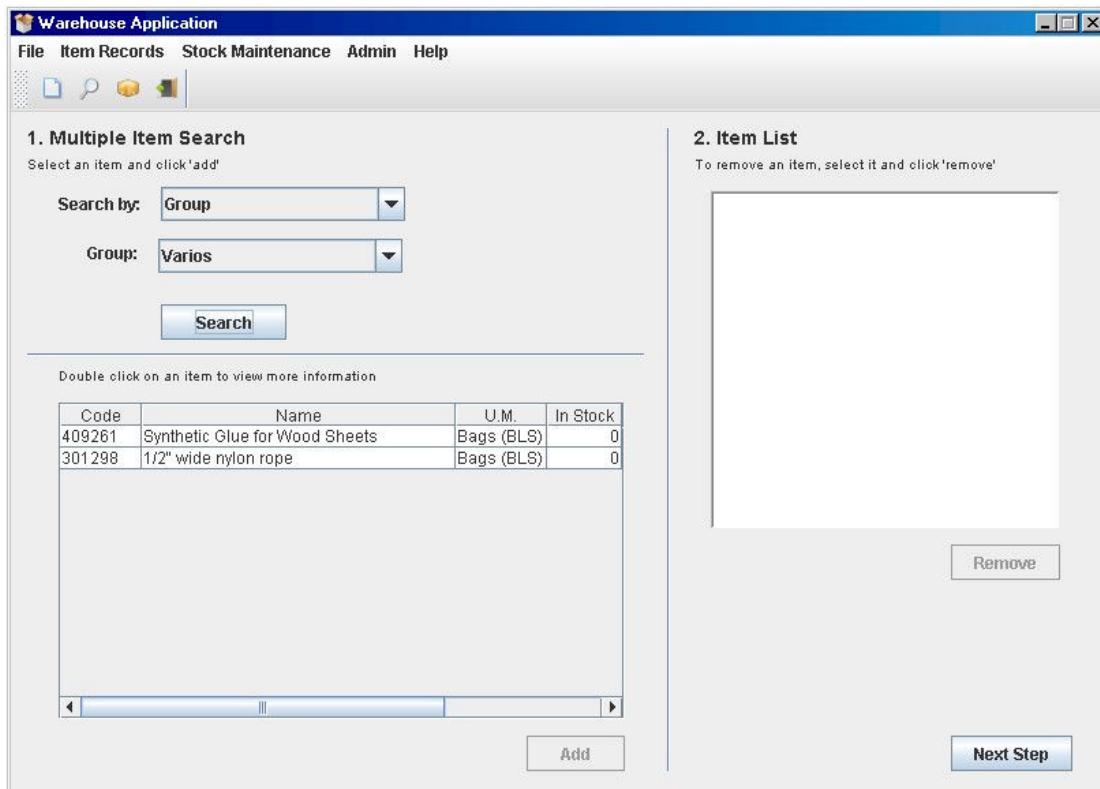
Search by: Group  
Group: Varios

The 'Search' button is pressed



*Result*

The appropriate results are displayed.

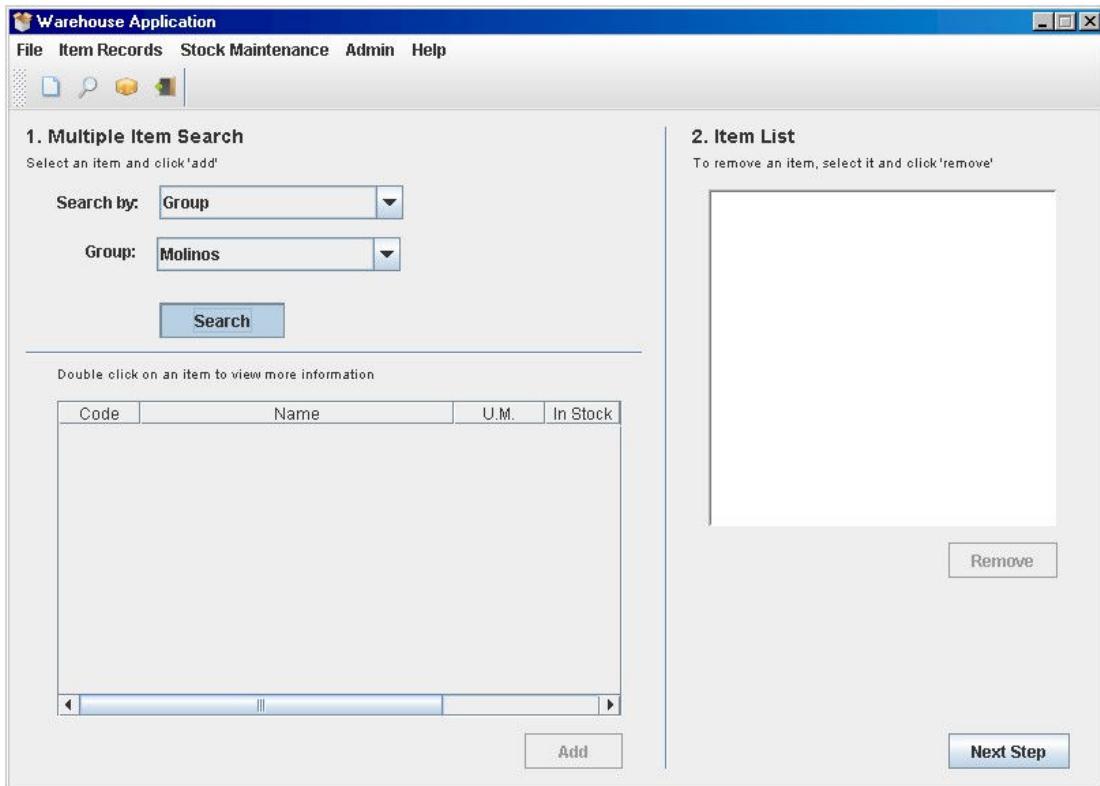


## 2. Searching for an item that does not exist

*Test*

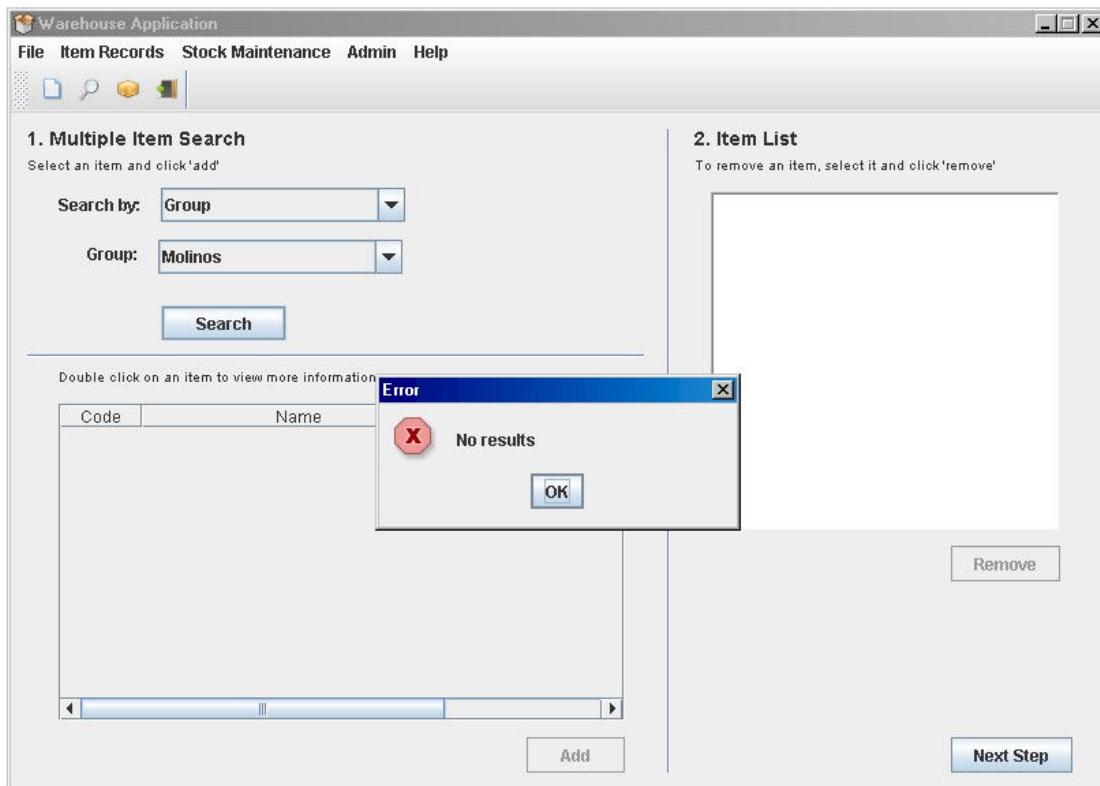
Search by: Group  
Group: Varios

The 'Search' button is pressed



### Result

No results message displayed



## SEARCHING FOR ONE OR MORE ITEMS BY LOCATION

Select Location from the drop-down list and the location search boxes load

1. Searching for an item that exists

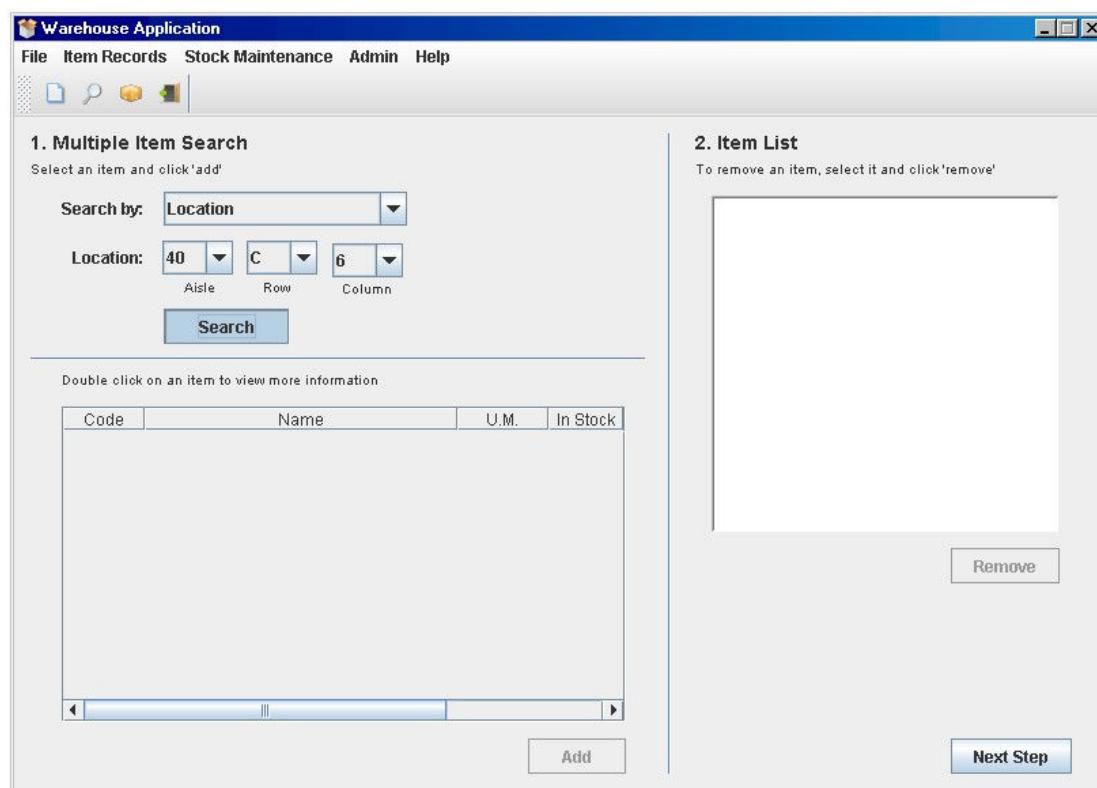
*Test*

Search by: Location

Location:

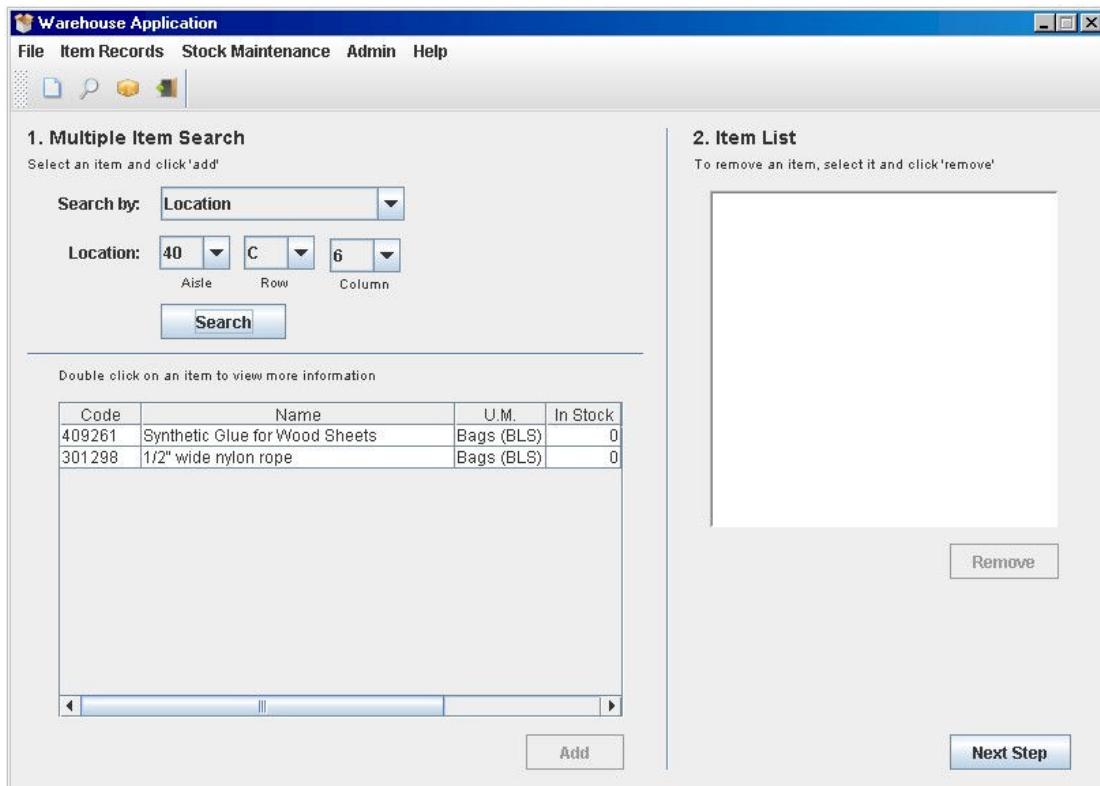
Aisle: 40  
Row: C  
Column: 6

The 'Search' button is pressed



*Result:*

Appropriate results are displayed

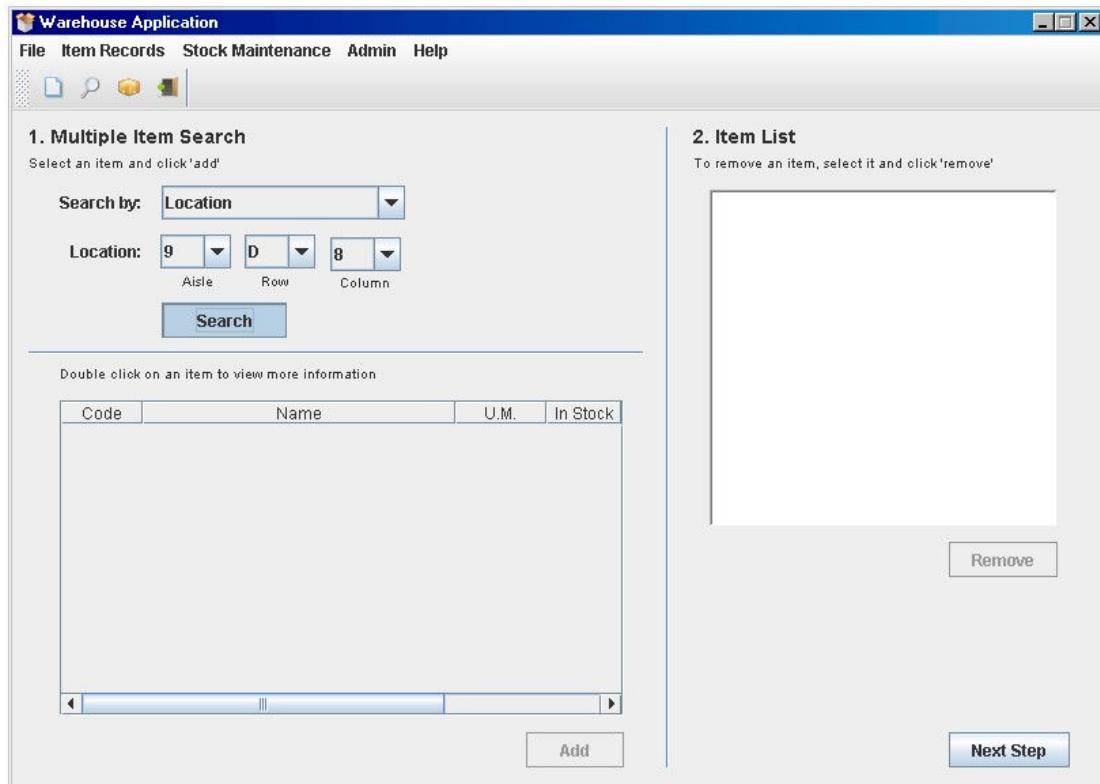


## 2. Searching for an item that does not exist

*Test*

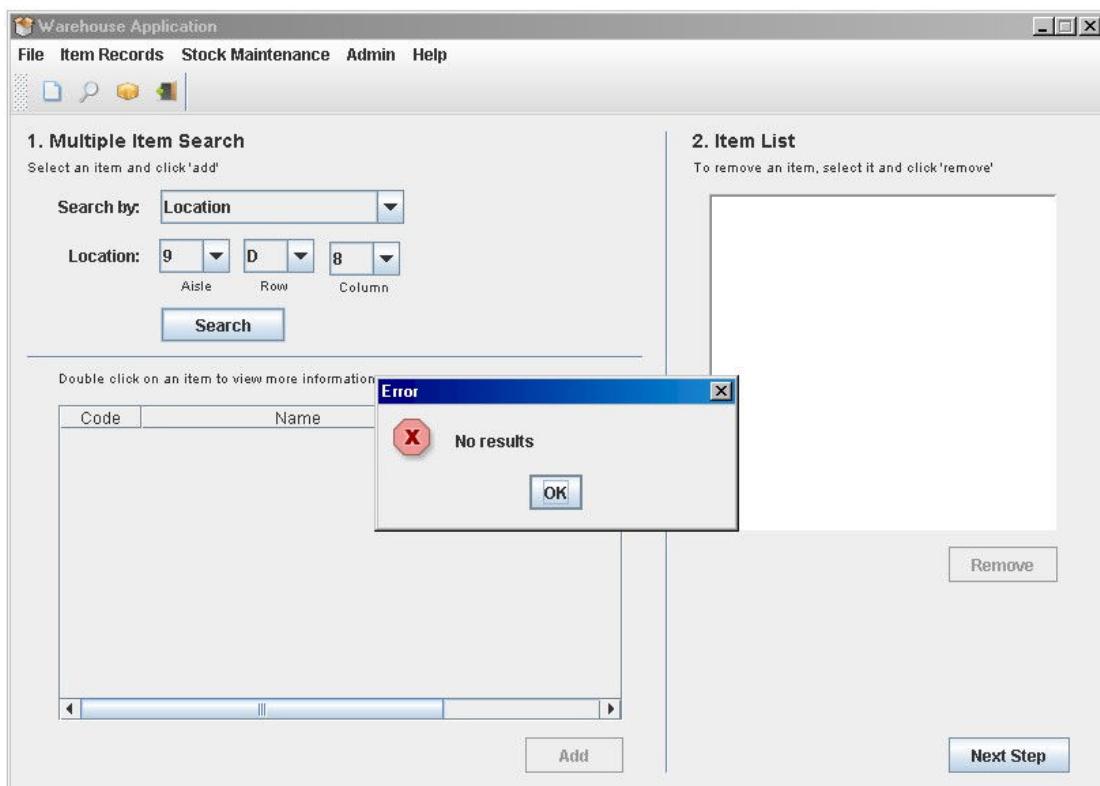
Search by: Location  
Location:  
Aisle: 9  
Row: D  
Column: 8

The 'Search' button is pressed



### Result

No results message displayed



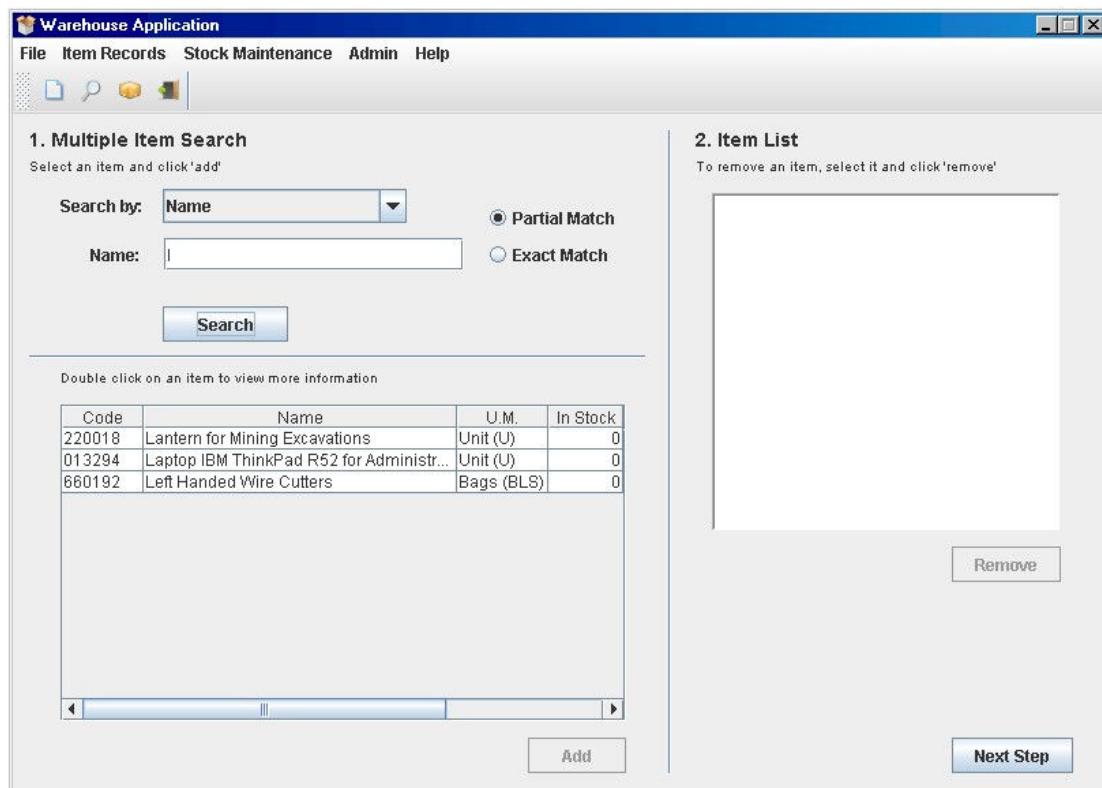
## ADDING ONE OR MORE ITEMS TO THE LIST OF ITEMS TO PROCESS

From the Multiple Item Search screen, applicable to both Item Entry and Exit operations

### CORRECT WAY

First, search for an item by any criteria, as described above

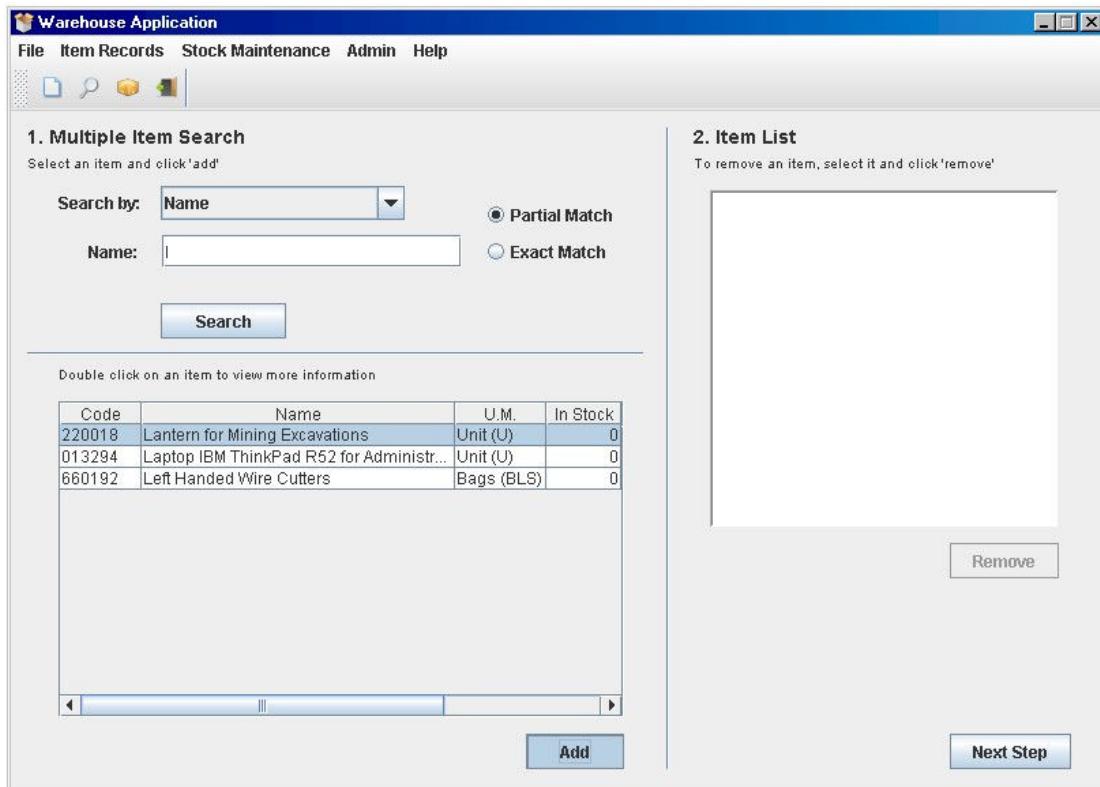
The item 'Lantern for Mining Excavations' will be added to the list



Note that as no row from the table is selected, the 'Add' button is not enabled (error handling)

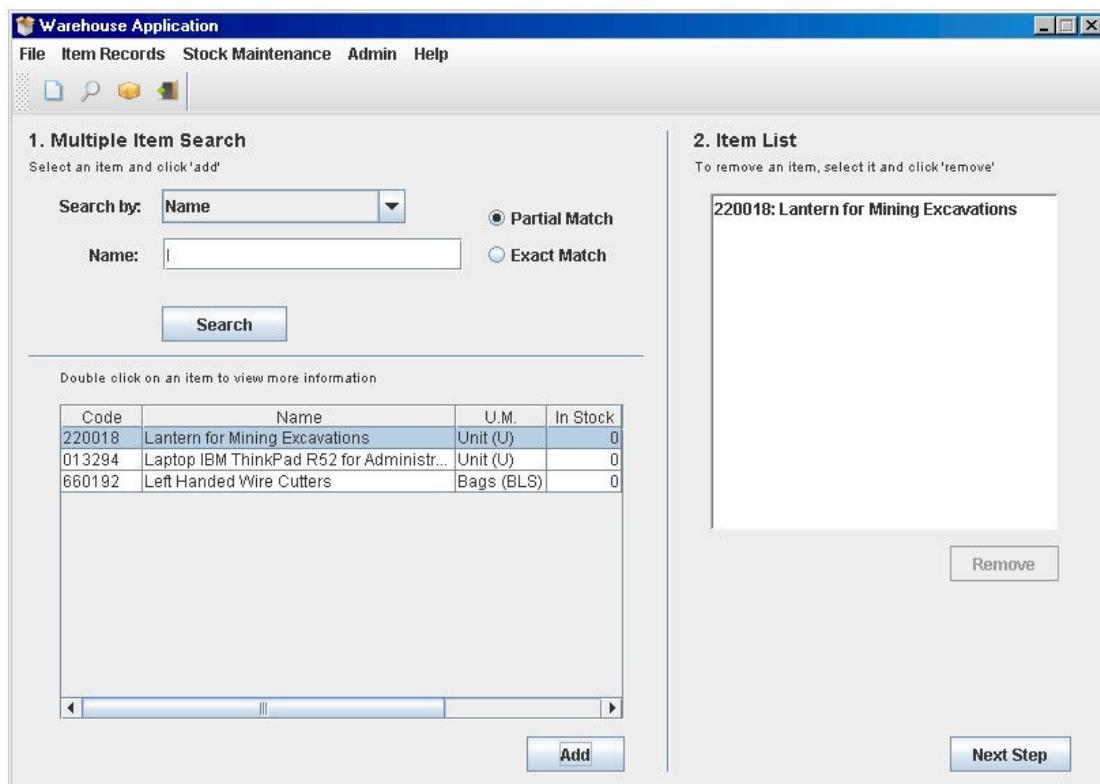
Select the item by clicking on the row

The 'Add' button is pressed



## Result

The item is added to the Item List



## INCORRECT WAY

Attempting to select multiple rows, by clicking and dragging from 'Lantern...' to 'Left handed...'

*Result*

Multiple row selections are not allowed. Only one item is selected

The screenshot shows a Windows application window titled 'Warehouse Application'. The menu bar includes 'File', 'Item Records', 'Stock Maintenance', 'Admin', and 'Help'. The toolbar has icons for New, Open, Save, Print, and Exit.

**1. Multiple Item Search:** A search interface with a dropdown 'Search by:' set to 'Name', a text input 'Name:' containing 'L', and two radio buttons for 'Partial Match' (selected) and 'Exact Match'. A 'Search' button is present. Below the search area is a note: 'Double click on an item to view more information'.

**2. Item List:** A list area with a note: 'To remove an item, select it and click 'remove''. It contains a single item: 'Left Handed Wire Cutters' (Code: 660192). A 'Remove' button is located below the list.

A table at the bottom left shows item details:

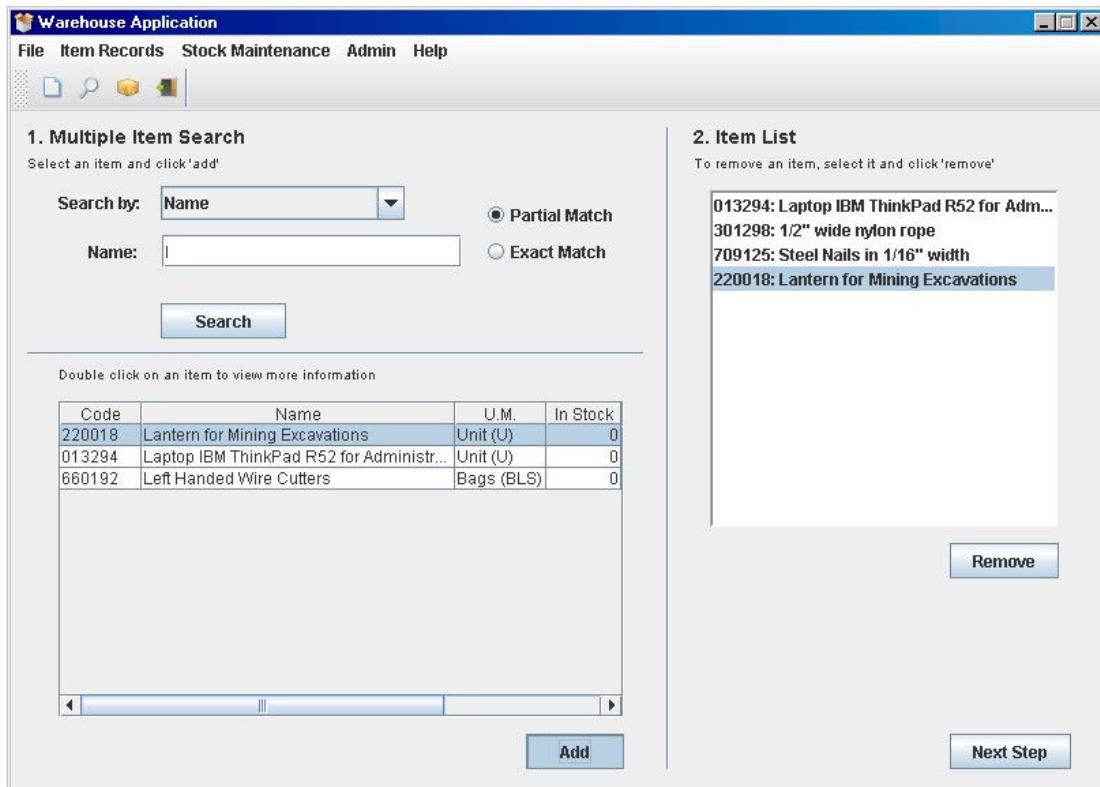
Code	Name	U.M.	In Stock
220018	Lantern for Mining Excavations	Unit (U)	0
013294	Laptop IBM ThinkPad R52 for Administr...	Unit (U)	0
660192	Left Handed Wire Cutters	Bags (BLS)	0

At the bottom right are 'Add' and 'Next Step' buttons.

Attempting to add an item that is already in the Item List

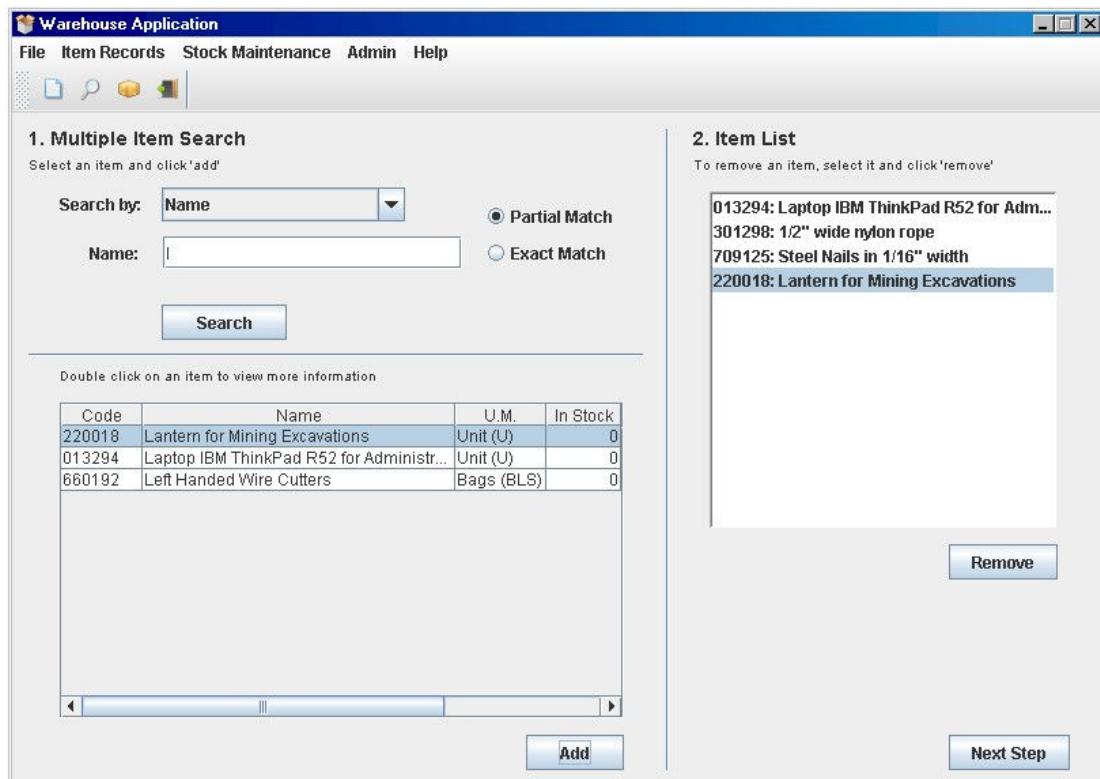
Adding the item 'Lantern for Mining Excavations' to a list that already contains it

The item is found, selected and the 'Add' button is pressed



## Result

The item is not added again to the list. No change



The Multiple Item Search screen is loaded for an Item Withdrawal transaction. A user attempts to add to the Items to Process list an item that is not currently in stock (i.e. withdraw an item that is not in the warehouse)

This is *only* applicable for Item Exit

Adding the item 'Synthetic Glue for Wood Sheets' item that is not in stock  
The item is found, selected and the 'Add' button is pressed

The screenshot shows the 'Warehouse Application' window divided into two main sections:

**1. Multiple Item Search:** This section contains a search interface. It includes a dropdown menu 'Search by:' set to 'Name', a text input field 'Name:' containing 'S', and two radio buttons for 'Partial Match' (selected) and 'Exact Match'. Below these are a 'Search' button and a note: 'Double click on an item to view more information'. A table lists items:

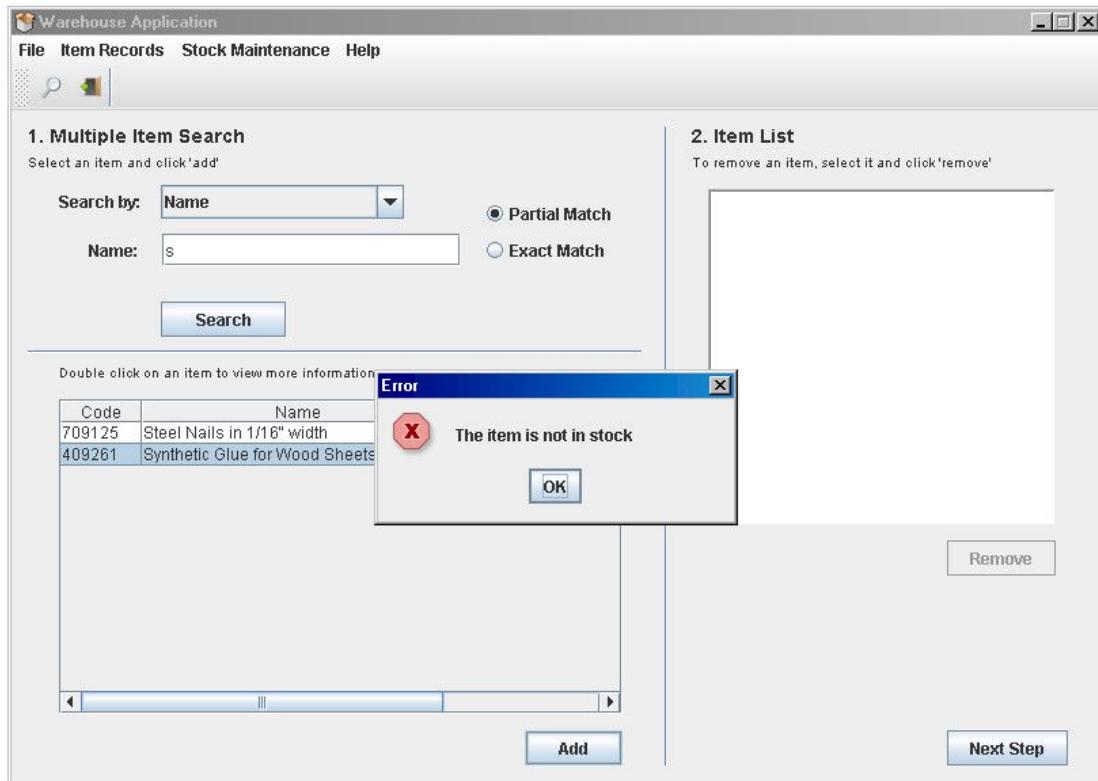
Code	Name	U.M.	In Stock
709125	Steel Nails in 1/16" width	Bags (BLS)	0
409261	Synthetic Glue for Wood Sheets	Bags (BLS)	0

**2. Item List:** This section contains a note: 'To remove an item, select it and click "remove"'. It features a large empty rectangular area for displaying items and a 'Remove' button below it. At the bottom right of this section is a 'Next Step' button.

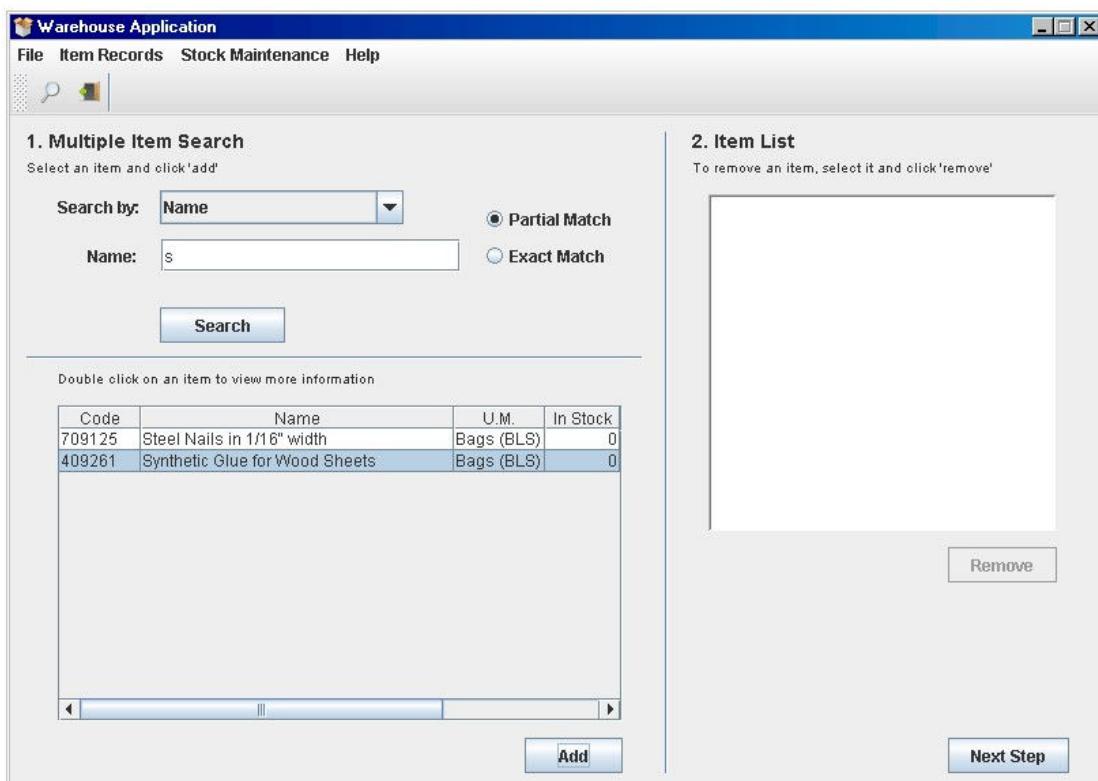
**Add**

*Result*

User is informed that the item is not in stock



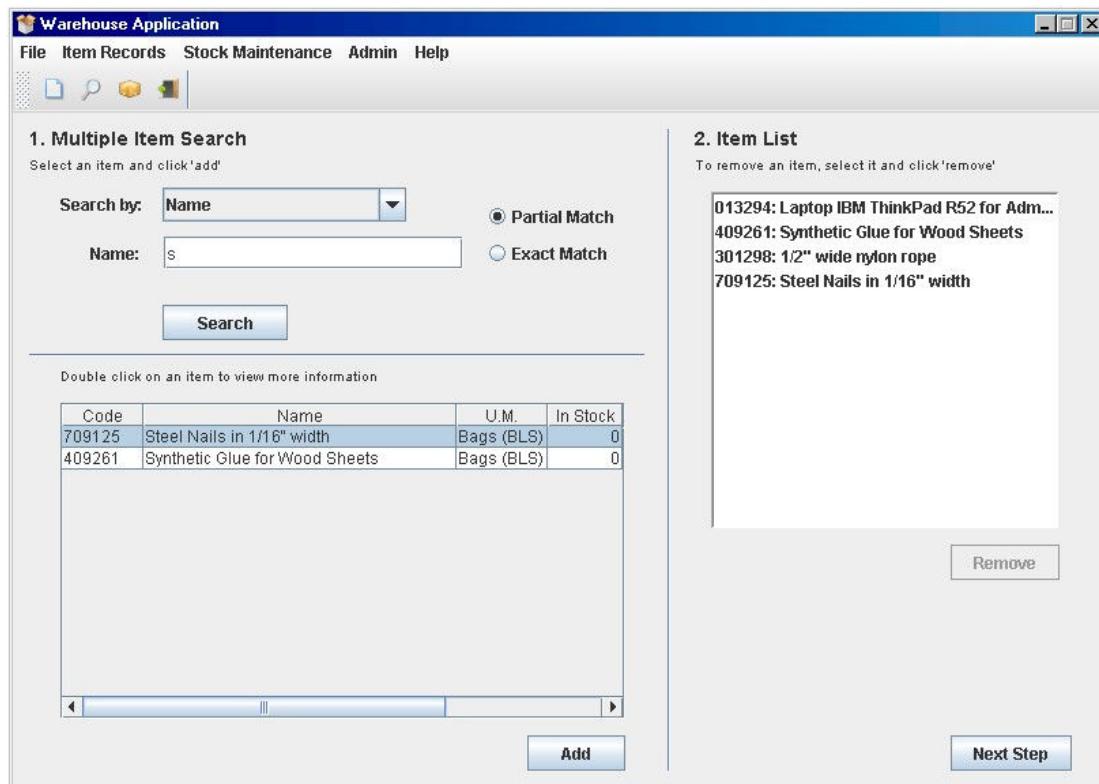
The item is not added to the Item List



## REMOVING ONE OR MORE ITEMS FROM THE LIST OF ITEMS TO PROCESS

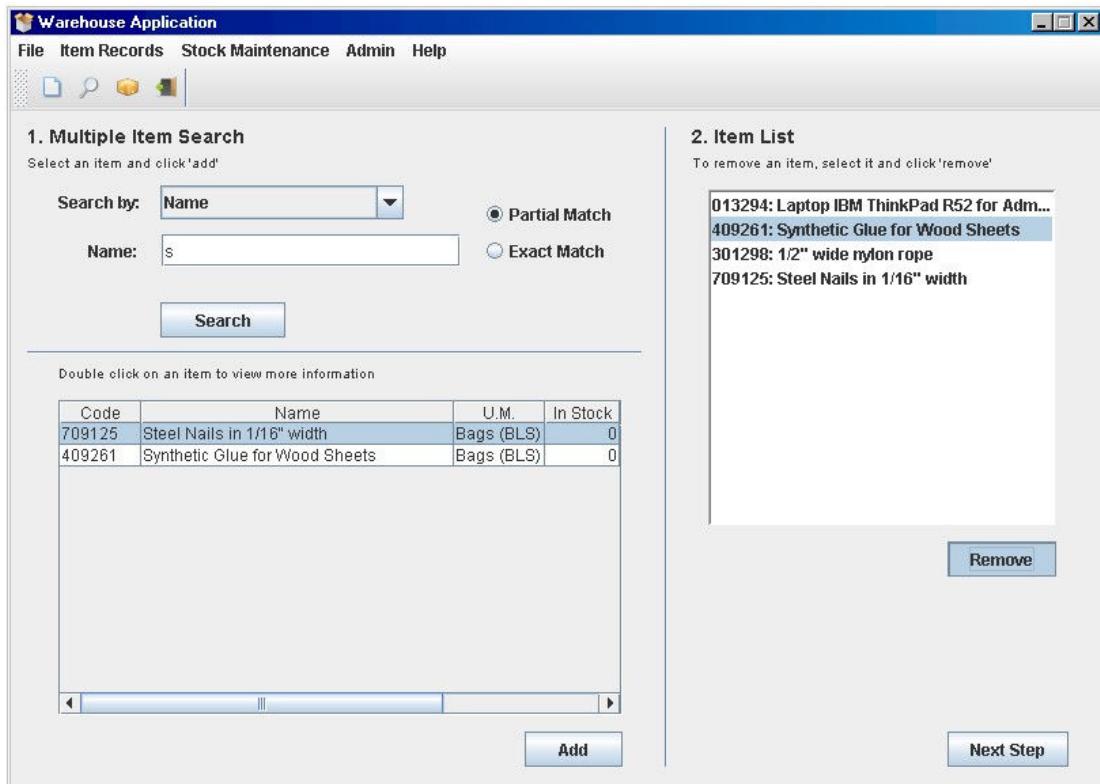
From the Multiple Item Search screen, applicable to both Item Entry and Exit operations

Remove the item 'Synthetic Glue for Wood Sheets' from the list:



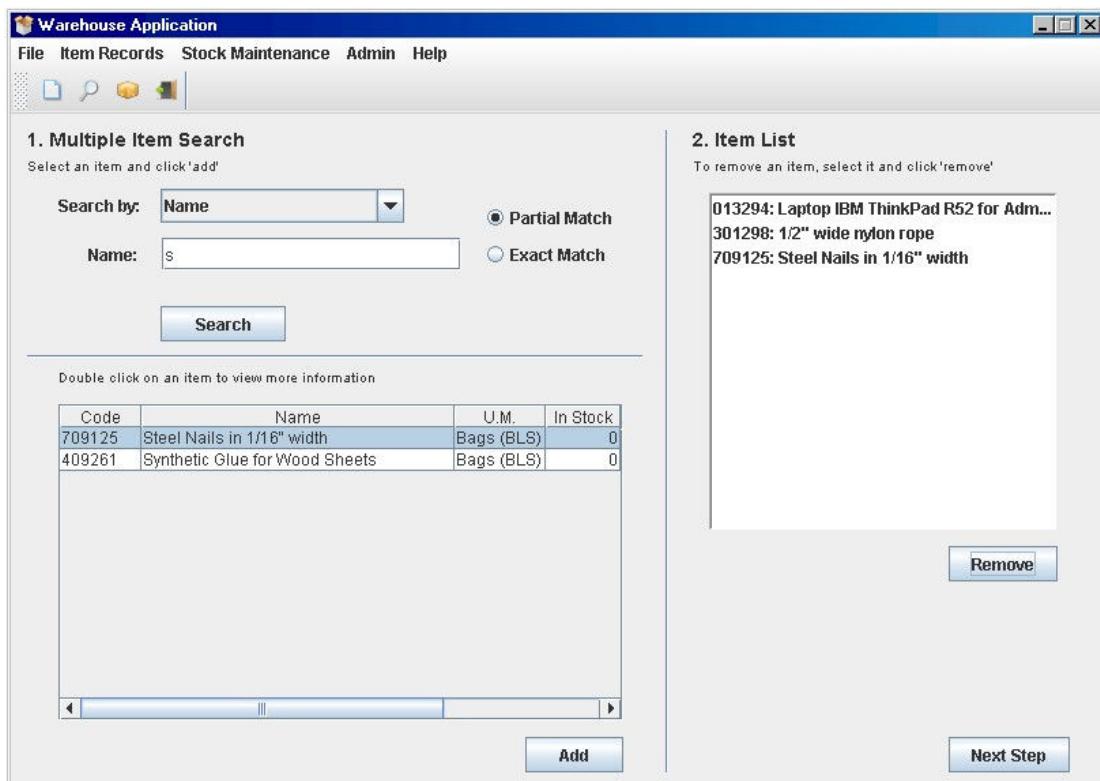
Note that when no item in the Item List is selected, the 'Remove' button is not enabled

Select the item by clicking on the row  
The 'Remove' button is pressed



## Result

The item is removed from the Item List

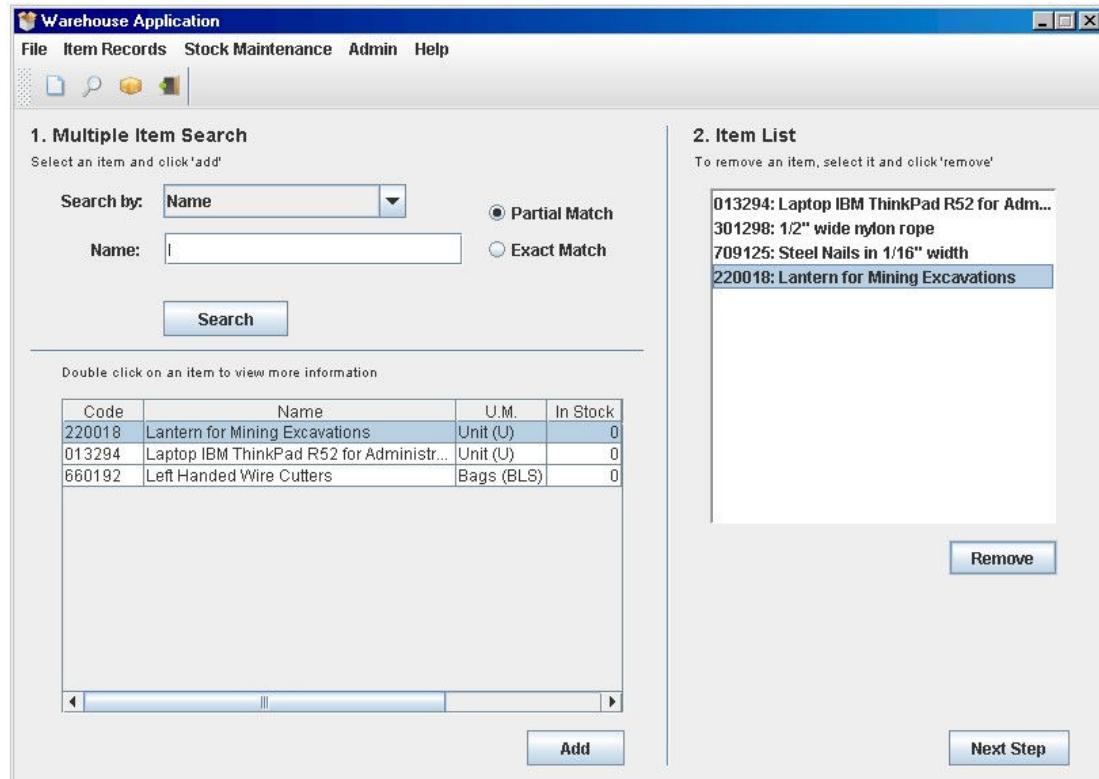


## INCORRECT WAY

Attempting to select multiple items, by clicking and dragging from 'Laptop...' to 'Steel...'

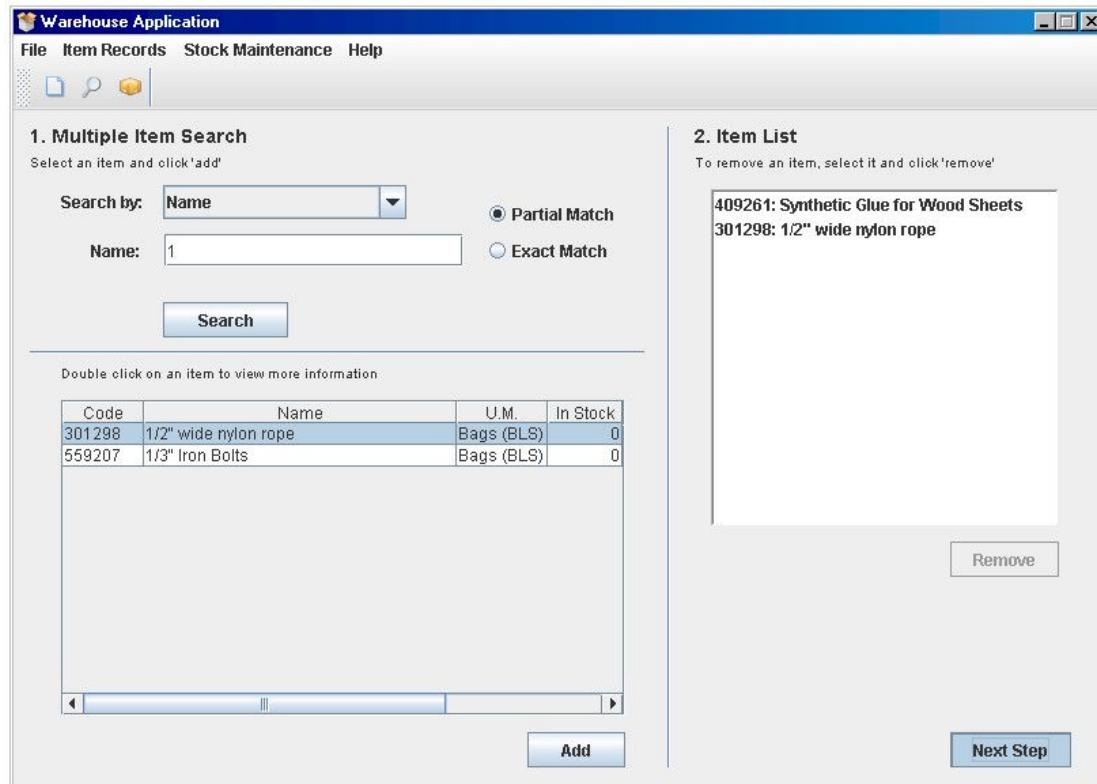
### *Result*

Multiple row selection is not allowed. Only one item is selected



## PROCESSING AN ITEM ENTRY TRANSACTION

A user searches for the items to process and adds them to the Item List. These items will be entered into the warehouse



The 'Next Step' button is pressed

*Result*

The Item Entry screen loads, with the items to process added to the table. Note that the items' unit of measurements are displayed, and the quantity column to be filled is the quantity of that unit of measurement that will be entered into the warehouse.

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Entry  
Fields marked with an \* are required

Proof of Reception No.\*:   
Received By: entryGuy

Delivered By\*:   
Date: 7/8/2010

5. Quantities  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (BLS)	0	0
301298	1/2" wide nylon rope	Metres (M)	0	0

**Finish**

### CORRECT INPUT

All fields filled and correct quantities entered

Proof of Reception No.: 21330012

Delivered by: Nexia Solutions S.A.C.

Quantities:

'Synthetic Glue for Wooden Sheets' 2  
'1/2" wide nylon rope' 100

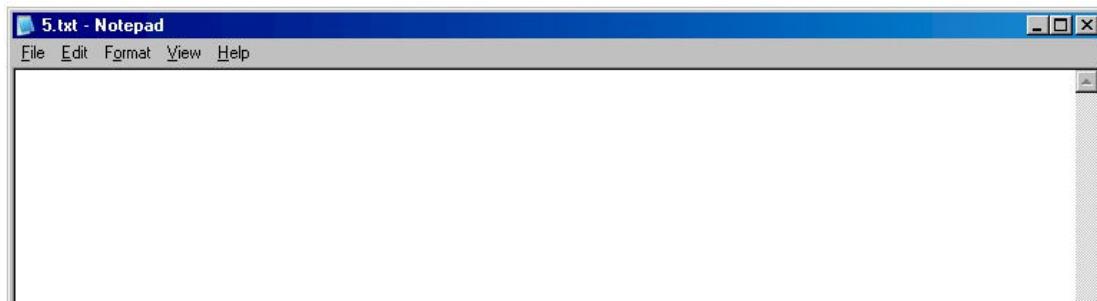
Transactions file BEFORE item entry transaction processed

For 'Synthetic Glue for Wooden Sheets' (ID: 4)

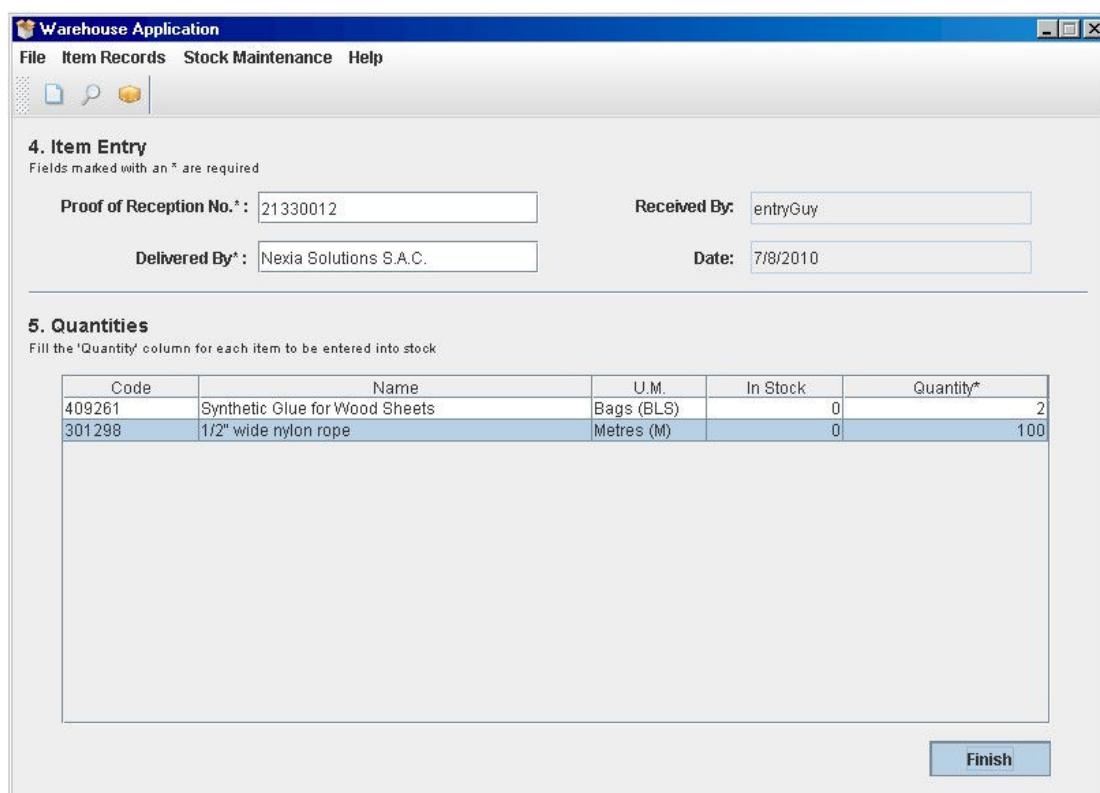
Transactions file empty



For '1/2" wide nylon rope' (ID: 5)  
Transactions file empty



The 'Finish' button is pressed



**4. Item Entry**  
Fields marked with an \* are required

Proof of Reception No.*:	21330012	Received By:	entryGuy
Delivered By*:	Nexia Solutions S.A.C.	Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (BLS)	0	2
301298	1/2" wide nylon rope	Metres (M)	0	100

**Finish**

*Result*

User informed that transaction has been successfully processed

Warehouse Application

File Item Records Stock Maintenance Help

4. Item Entry  
Fields marked with an \* are required

Proof of Reception No.\*: 21330012 Received By: entryGuy

Delivered By\*: Nexia Solutions S.A.C. Date: 7/8/2010

5. Quantities  
Fill the 'Quantity' column for each item to be entered into stock

Code	Description	In Stock	Quantity*
409261	Synthetic Glue for Wood	0	2
301298	1/2" wide nylon rope	0	100

Success  
Transactions successfully processed  
OK

Finish

Transactions file AFTER item entry transaction processed

For 'Synthetic Glue for Wooden Sheets' (ID: 4)  
Transaction created

4.txt - Notepad

File Edit Format View Help

```
0|0|7|8|2010|2|2|entryGuy|21330012|Nexia solutions S.A.C.
```

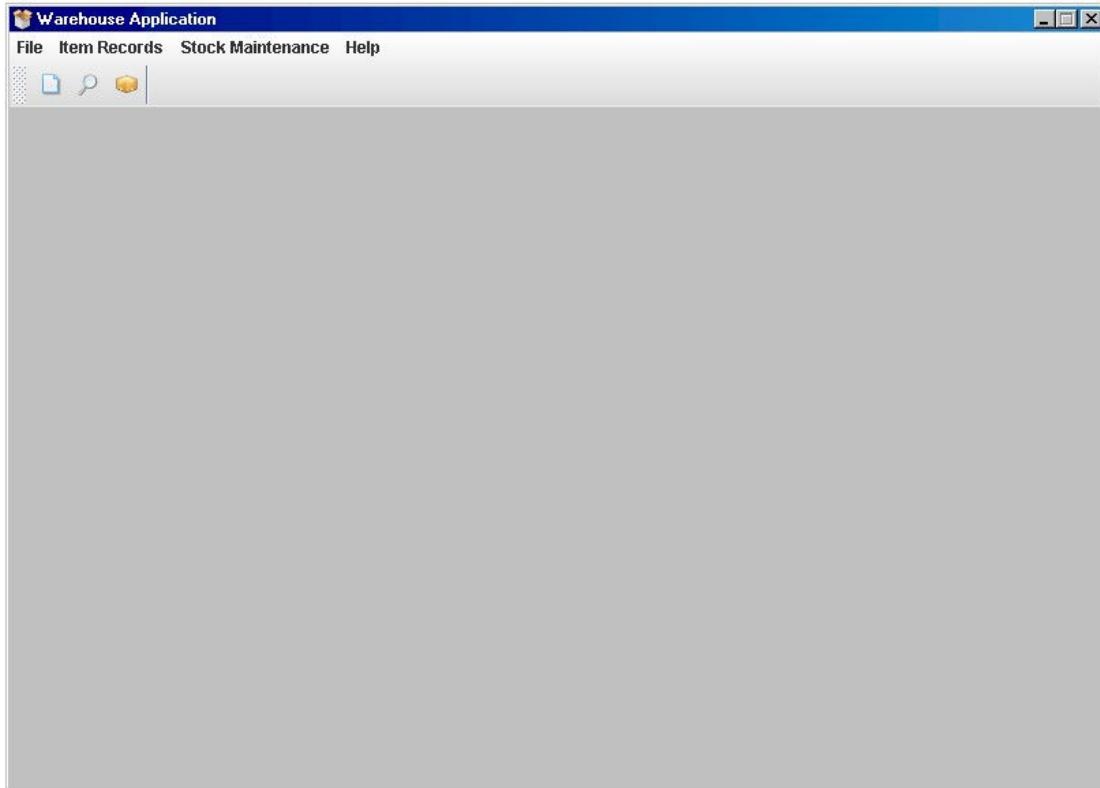
For '1/2" wide nylon rope' (ID: 5)  
Transactions created

5.txt - Notepad

File Edit Format View Help

```
0|0|7|8|2010|100|100|entryGuy|21330012|Nexia solutions S.A.C.
```

Application returns to the landing screen



### **INCORRECT INPUT**

1. Some fields left blank

Proof of Reception No.: 32132132  
Delivered by: [empty]  
Quantities:  
'Synthetic Glue for Wooden Sheets' 1  
'1/2" wide nylon rope' 2

The 'Finish' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Entry**  
Fields marked with an \* are required

Proof of Reception No.*:	32132132	Received By:	entryGuy
Delivered By*:		Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (BL...)	2	1
301298	1/2" wide nylon rope	Metres (M)	100	2

**Finish**

### Result

The user is informed that some fields are missing

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Entry**  
Fields marked with an \* are required

Proof of Reception No.*:	32132132	Received By:	entryGuy
Delivered By*:		Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	Stock	Quantity*
409261	Synthetic Glue for Wood	2	1
301298	1/2" wide nylon rope	100	2

An error dialog box is displayed in the center of the screen, showing the message "Some fields are missing".

**Finish**

2. Not all quantities filled in (left at 0)

Proof of Reception No.: 32132132  
Delivered by: Nexia Solutions S.A.C.  
Quantities:

'Synthetic Glue for Wooden Sheets'	0	[invalid]
'1/2" wide nylon rope'	2	

The 'Finish' button is pressed

The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", and "Help". Below the menu is a toolbar with icons for file operations. The main area is divided into sections:

- 4. Item Entry:** Fields include "Proof of Reception No.\*" (21330012), "Received By" (entryGuy), "Delivered By" (Nexia Solutions S.A.C.), and "Date" (7/8/2010).
- 5. Quantities:** A table for entering quantities for items received. The table has columns: Code, Name, U.M., In Stock, and Quantity\*. The data is as follows:

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (BLS)	0	2
301298	1/2" wide nylon rope	Metres (M)	0	100

A "Finish" button is located at the bottom right of the form area.

*Result*

User informed that some fields are missing. Transaction does not proceed

Warehouse Application

File Item Records Stock Maintenance Help

4. Item Entry  
Fields marked with an \* are required

Proof of Reception No.\*: 32132132      Received By: entryGuy

Delivered By\*: Nexia Solutions S.A.C.      Date: 7/8/2010

5. Quantities  
Fill the 'Quantity' column for each item to be entered into stock

Code	Description	Stock	Quantity*
409261	Synthetic Glue for Wood	2	0
301298	1/2" wide nylon rope	100	2

Error  
Some fields are missing or incorrect  
OK

Finish

3. A quantity less than 0

Proof of Reception No.: 32132132

Delivered by: Nexia Solutions S.A.C.

Quantities:

'Synthetic Glue for Wooden Sheets'	3
'1/2" wide nylon rope'	-6 [invalid]

The 'Finish' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Entry**  
Fields marked with an \* are required

Proof of Reception No.*:	32132132	Received By:	entryGuy
Delivered By*:	Nexia Solutions S.A.C.	Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags...	2	3
301298	1/2" wide nylon rope	Metres...	100	-6

**Finish**

### Result

The user is informed that some fields are missing or incorrect. The transaction operation does not follow through

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Entry**  
Fields marked with an \* are required

Proof of Reception No.*:	32132132	Received By:	entryGuy
Delivered By*:	Nexia Solutions S.A.C.	Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	Stock	Quantity*
409261	Synthetic Glue for Woo	2	3
301298	1/2" wide nylon rope	100	-6

**Error**

Some fields are missing or incorrect

**OK**

**Finish**

4. A non-integer quantity (containing decimals)

Proof of Reception No.: 32132132

Delivered by: Nexia Solutions S.A.C.

Quantities:

'Synthetic Glue for Wooden Sheets'	1.54	[invalid]
'1/2" wide nylon rope'	2	

The quantity is entered and Return is hit

*Result*

The field highlights in red to show the user that the input is invalid. The user must change it before transaction to proceed.

The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", and "Help". Below the menu is a toolbar with icons for New, Search, and Open. The main area has two sections: "4. Item Entry" and "5. Quantities".

**4. Item Entry**  
Fields marked with an \* are required.

Proof of Reception No.*:	32132132	Received By:	entryGuy
Delivered By*:	Nexia Solutions S.A.C.	Date:	7/8/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be entered into stock.

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (...)	2	1.54
301298	1/2" wide nylon rope	Metres...	100	2

A "Finish" button is located at the bottom right of the form.

5. A non-integer quantity (containing characters)

Proof of Reception No.: 32132132

Delivered by: Nexia Solutions S.A.C.

Quantities:

'Synthetic Glue for Wooden Sheets'	4	
'1/2" wide nylon rope'	ab73	[invalid]

The quantity is entered and Return is hit

*Result*

The field highlights in red to show the user that the input is invalid. The user must change it before transaction to proceed.

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Entry  
Fields marked with an \* are required

Proof of Reception No.\*:  Received By:

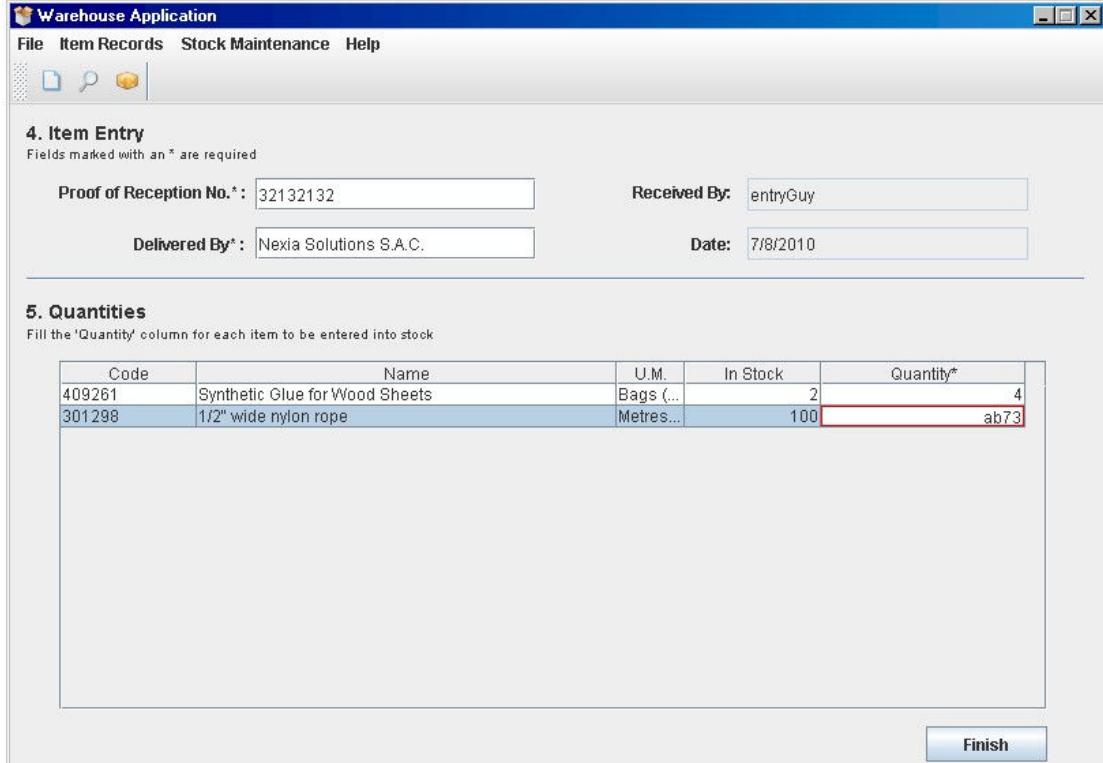
Delivered By:

Date:

5. Quantities  
Fill the 'Quantity' column for each item to be entered into stock

Code	Name	U.M.	In Stock	Quantity*
409261	Synthetic Glue for Wood Sheets	Bags (...)	2	4
301298	1/2" wide nylon rope	Metres...	100	ab73

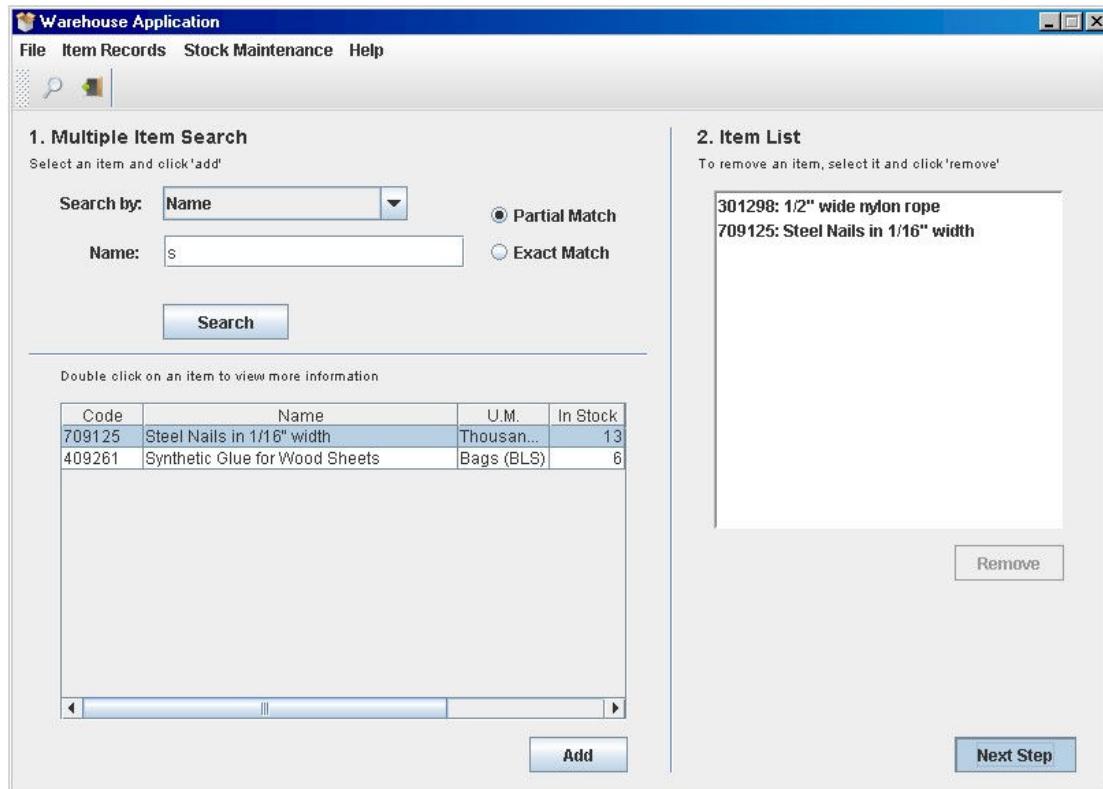
**Finish**



## PROCESSING AN ITEM EXIT TRANSACTION

A user searches for the items to process and adds them to the Item List. These items will be withdrawn from the warehouse

The 'Next Step' button is pressed



### Result

The Item Exit screen loads, with the items to process added to the table. Note that the items' unit of measurements are displayed, and the quantity column to be filled is the quantity of that unit of measurement that will be removed from the warehouse.

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.\*:   
Dispatched By: exitGuy  
Requested By\*:   
Date: 7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	102	0
709125	Steel Nails in 1/16" width	Thous...	13	0

**Next Step**

## CORRECT INPUT

All fields filled and correct quantities entered

Item Exit Voucher No.: 00129821  
Requested by: A. Morales  
Quantities:  
'1/2" wide nylon rope' 50  
'Steel nails in 1/16" width' 3

Transactions file BEFORE item exit transaction processed

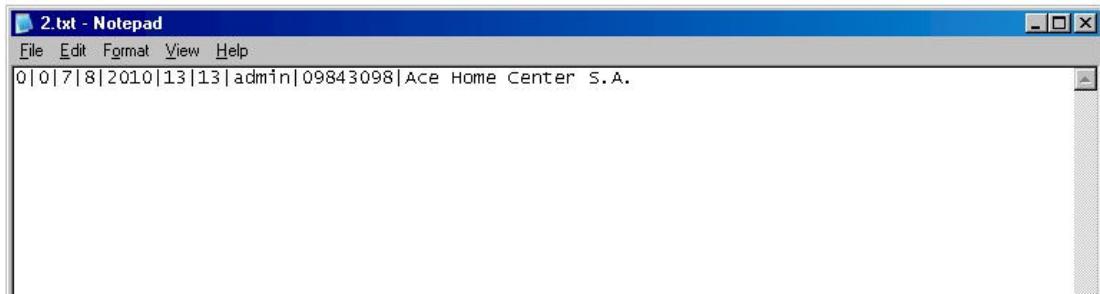
For '1/2" wide nylon rope' (ID: 5)

5.txt - Notepad

File Edit Format View Help

```
0|0|7|8|2010|100|100|entryGuy|21330012|Nexia Solutions S.A.C.  
1|0|7|8|2010|2|102|entryGuy|32132132|Nexia Solutions S.A.C.
```

For 'Steel nails in 1/16" width' (ID: 2)



The 'Finish' button is pressed

The screenshot shows the "Warehouse Application" window. The menu bar includes File, Item Records, Stock Maintenance, and Help. The main area is divided into sections:

- 4. Item Exit:** Fields include "Item Exit Voucher No.\*: 00129821", "Dispatched By: exitGuy", "Requested By: A. Morales", and "Date: 7/7/2010".
- 5. Quantities:** A table lists items with their current stock levels and quantities to be removed:

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	102	50
709125	Steel Nails in 1/16" width	Thous...	13	3

A "Next Step" button is located at the bottom right of the form.

*Result*

The user is informed that the operation has been successful

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.*:	00129821	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	Stock	Quantity*
301298	1/2" wide nylon rope	102	50
709125	Steel Nails in 1/16" width	13	3

**Success**  
Transactions successfully processed

OK

Next Step

The optimal route screen loads [This to be described later]

**Warehouse Application**

File Item Records Stock Maintenance Help

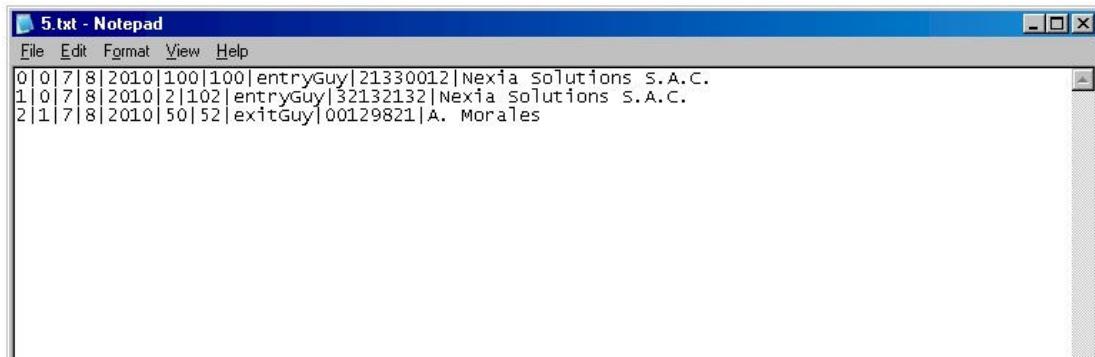
6. Optimal Route  
The column 'Order' denotes the pickup route

Order	Code	Name	UM	Location
1	709125	Steel Nails in 1/16" width	Thousands (MLL)	A01.28.A05
2	301298	1/2" wide nylon rope	Metres (M)	A01.40.C06

Finish

Transactions file AFTER item exit transaction processed

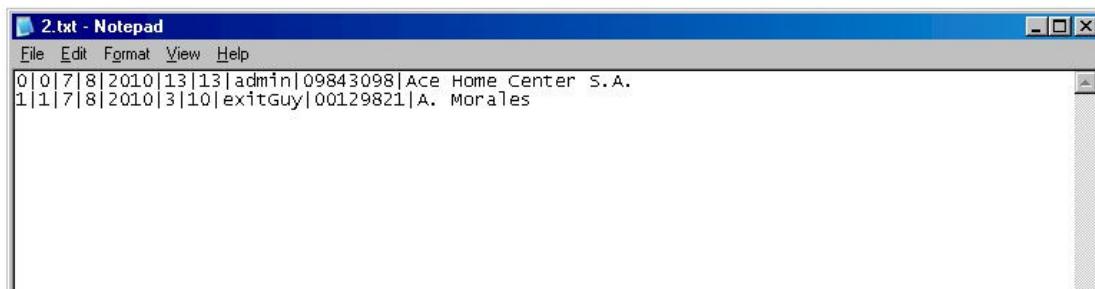
For '1/2" wide nylon rope' (ID: 5)  
Transaction created



5.txt - Notepad

```
File Edit Format View Help
0|0|7|8|2010|100|100|entryGuy|21330012|Nexia Solutions S.A.C.
1|0|7|8|2010|2|102|entryGuy|32132132|Nexia Solutions S.A.C.
2|1|7|8|2010|50|52|exitGuy|00129821|A. Morales
```

For 'Steel nails in 1/16" width' (ID: 2)  
Transaction created



2.txt - Notepad

```
File Edit Format View Help
0|0|7|8|2010|13|13|admin|09843098|Ace Home Center S.A.
1|1|7|8|2010|3|10|exitGuy|00129821|A. Morales
```

### INCORRECT INPUT

1. Some fields left blank

Item Exit Voucher No.: [empty]  
Requested by: A. Morales  
Quantities:  
    '1/2" wide nylon rope'                  1  
    'Steel nails in 1/16" width'              3

The 'Next Step' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.*:	<input type="text"/>	Dispatched By:	<input type="text"/> exitGuy
Requested By*:	<input type="text"/> A. Morales	Date:	<input type="text"/> 7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	52	1
709125	Steel Nails in 1/16" width	Thous...	10	3

**Next Step**

### Result

User is informed that some fields are missing

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.*:	<input type="text"/>	Dispatched By:	<input type="text"/> exitGuy
Requested By*:	<input type="text"/> A. Morales	Date:	<input type="text"/> 7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	Stock	Quantity*
301298	1/2" wide nylon rope	52	1
709125	Steel Nails in 1/16" width	10	3

**Error**  
Some fields are missing

**OK**

**Next Step**

2. Not all quantities filled in (left at 0)

Item Exit Voucher No.: 80381129  
Requested by: A. Morales  
Quantities:

'1/2" wide nylon rope'	0
'Steel nails in 1/16" width'	3

The 'Next Step' button is pressed

The screenshot shows a Windows application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", and "Help". Below the menu is a toolbar with icons for search and print. The main area is divided into sections:

- 4. Item Exit:** Fields include "Item Exit Voucher No.\*" (80381129), "Dispatched By" (exitGuy), "Requested By" (A. Morales), and "Date" (7/7/2010).
- 5. Quantities:** A table lists items and their current stock levels:

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	52	0
709125	Steel Nails in 1/16" width	Thous...	10	3

A "Next Step" button is located at the bottom right of the form.

*Result*

User informed that one or more quantities are incorrect

Warehouse Application

File Item Records Stock Maintenance Help

4. Item Exit

Fields marked with an \* are required

Item Exit Voucher No.*:	80381129	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

5. Quantities

Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Description	Quantity*
301298	1/2" wide nylon rope	52
709125	Steel Nails in 1/16	10

Error

One or more quantities entered are incorrect

OK

Next Step

3. A quantity greater than the quantity in stock is entered

Item Exit Voucher No.: 80381129

Requested by: A. Morales

In Stock:

'1/2" wide nylon rope'	10
'Steel nails in 1/16" width'	2

Quantities to be removed:

'1/2" wide nylon rope'	15
'Steel nails in 1/16" width'	3

The 'Next Step' button is pressed

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	83209840	Dispatched By:	admin
Requested By*:	A. Morales	Date:	25/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	10	15
709125	Steel Nails in 1/16" width	Thous...	2	10

**Next Step**

### Result

The user is informed that one or more quantities are incorrect

**Warehouse Application**

File Item Records Stock Maintenance Admin Help

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	83209840	Dispatched By:	admin
Requested By*:	A. Morales	Date:	25/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	10	15
709125	Steel Nails in 1/16" width	Thous...	2	10

**Error**

One or more quantities entered are incorrect

**OK**

**Next Step**

4. A quantity less than 0

Item Exit Voucher No.: 80381129  
Requested by: A. Morales  
Quantities:  
'1/2" wide nylon rope' -8 [invalid]  
'Steel nails in 1/16" width' 3

The 'Next Step' button is pressed

Warehouse Application

File Item Records Stock Maintenance Help

4. Item Exit

Fields marked with an \* are required

Item Exit Voucher No.*:	80381129	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

5. Quantities

Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	52	-8
709125	Steel Nails in 1/16" width	Thous...	10	3

Next Step

*Result*

User informed that one or more quantities entered are incorrect

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	80381129	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Description	Quantity*
301298	1/2" wide nylon rope	52
709125	Steel Nails in 1/16	10

**Error**  
One or more quantities entered are incorrect

**OK**

**Next Step**

### 5. A non-integer quantity (containing decimals)

Item Exit Voucher No.: 80381129  
Requested by: A. Morales  
Quantities:

'1/2" wide nylon rope'	1
'Steel nails in 1/16" width'	3.14 [invalid]

The quantity is entered and Return is hit

*Result*

The field highlights in red to show the user that the input is invalid. The user must change it before transaction to proceed.

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.*:	80381129	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	52	1
709125	Steel Nails in 1/16" width	Thous...	10	3.14

**Next Step**

6. A non-integer quantity (containing characters)

Item Exit Voucher No.: 80381129

Requested by: A. Morales

Quantities:

'1/2" wide nylon rope'	a2bc3 [invalid]
'Steel nails in 1/16" width'	3

The quantity is entered and Return is hit

*Result*

**Warehouse Application**

File Item Records Stock Maintenance Help

4. Item Exit  
Fields marked with an \* are required

Item Exit Voucher No.*:	80381129	Dispatched By:	exitGuy
Requested By*:	A. Morales	Date:	7/7/2010

5. Quantities  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
301298	1/2" wide nylon rope	Metres...	52	a2bc3
709125	Steel Nails in 1/16" width	Thous...	10	3

**Next Step**

The field highlights in red to show the user that the input is invalid. The user must change it before transaction to proceed.

## GETTING THE OPTIMAL PICKUP ROUTE

This is the last step of the Item Exit process.

A user searches for items and adds them to the Item List

The screenshot shows a software application window titled "Warehouse Application". The menu bar includes "File", "Item Records", "Stock Maintenance", and "Help".

**1. Multiple Item Search:** A search interface with the following fields:

- Search by:
- Partial Match
- Exact Match
- Name:
- 

Below the search area is a note: "Double click on an item to view more information". A table displays two items found:

Code	Name	U.M.	In Stock
301298	1/2" wide nylon rope	Metres (M)	50
559207	1/3" Iron Bolts	Bags (BLS)	15

**2. Item List:** A list of items currently selected for pickup:

- 220018: Lantern for Mining Excavations
- 013294: Laptop IBM ThinkPad R52 for Adm...
- 660192: Left Handed Wire Cutters
- 709125: Steel Nails in 1/16" width
- 409261: Synthetic Glue for Wood Sheets
- 301298: 1/2" wide nylon rope
- 559207: 1/3" Iron Bolts

Buttons for managing the list include "Remove" and "Add".

'Next Step' is pressed.

The fields are filled in and the correct quantities are entered

'Next Step' is pressed

[These steps have been detailed above]

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	09843329	Dispatched By:	exitGuy
Requested By*:	J. Peterman	Date:	7/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
220018	Lantern for Mining Excavations	Unit (U)	4	1
013294	Laptop IBM ThinkPad R52 for Administrative Use	Unit (U)	1	1
660192	Left Handed Wire Cutters	Bags (...)	12	10
709125	Steel Nails in 1/16" width	Thous...	7	3
409261	Synthetic Glue for Wood Sheets	Bags (...)	6	1
301298	1/2" wide nylon rope	Metres...	50	20
559207	1/3" Iron Bolts	Bags (...)	15	5

**Next Step**

## Result

User informed that transactions were processed

**Warehouse Application**

File Item Records Stock Maintenance Help

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	09843329	Dispatched By:	exitGuy
Requested By*:	J. Peterman	Date:	7/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

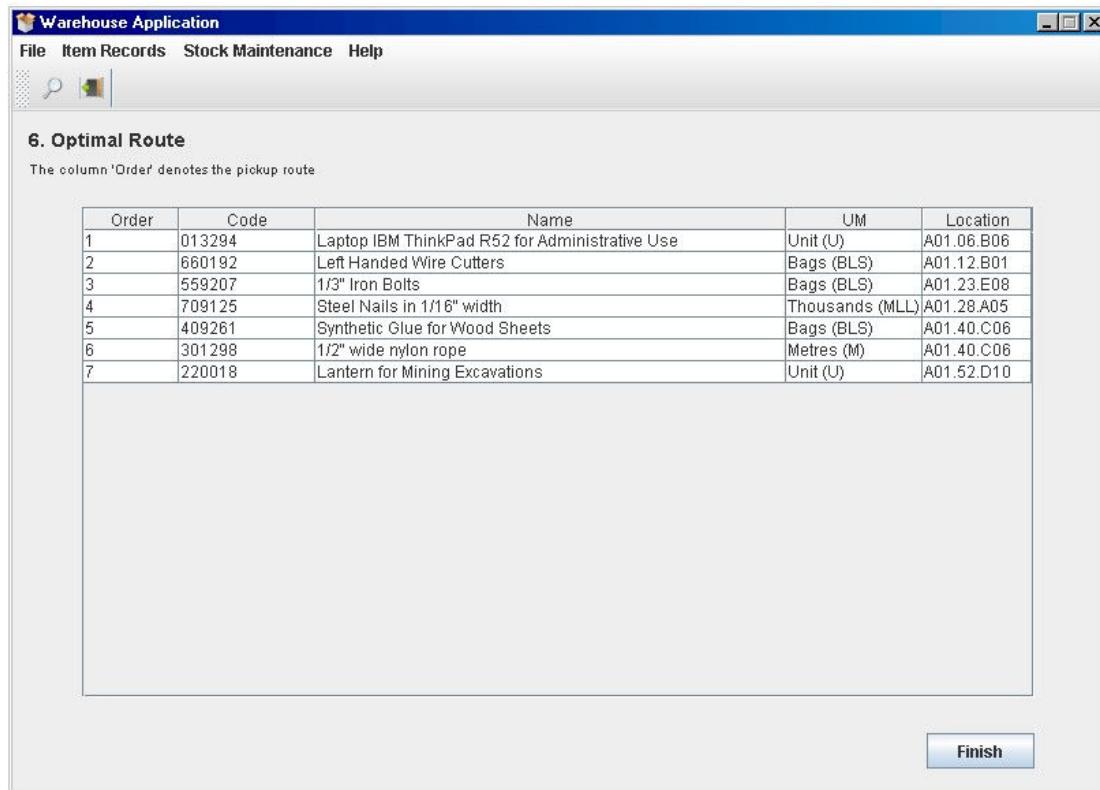
Code	Name	Stock	Quantity*
220018	Lantern for Mining Excavations	4	1
013294	Laptop IBM ThinkPad R52 for Administrative Use	1	1
660192	Left Handed Wire Cutters	12	10
709125	Steel Nails in 1/16" width	7	3
409261	Synthetic Glue for Wood Sheets	6	1
301298	1/2" wide nylon rope	50	20
559207	1/3" Iron Bolts	15	5

**Success**  
Transactions successfully processed

**OK**

**Next Step**

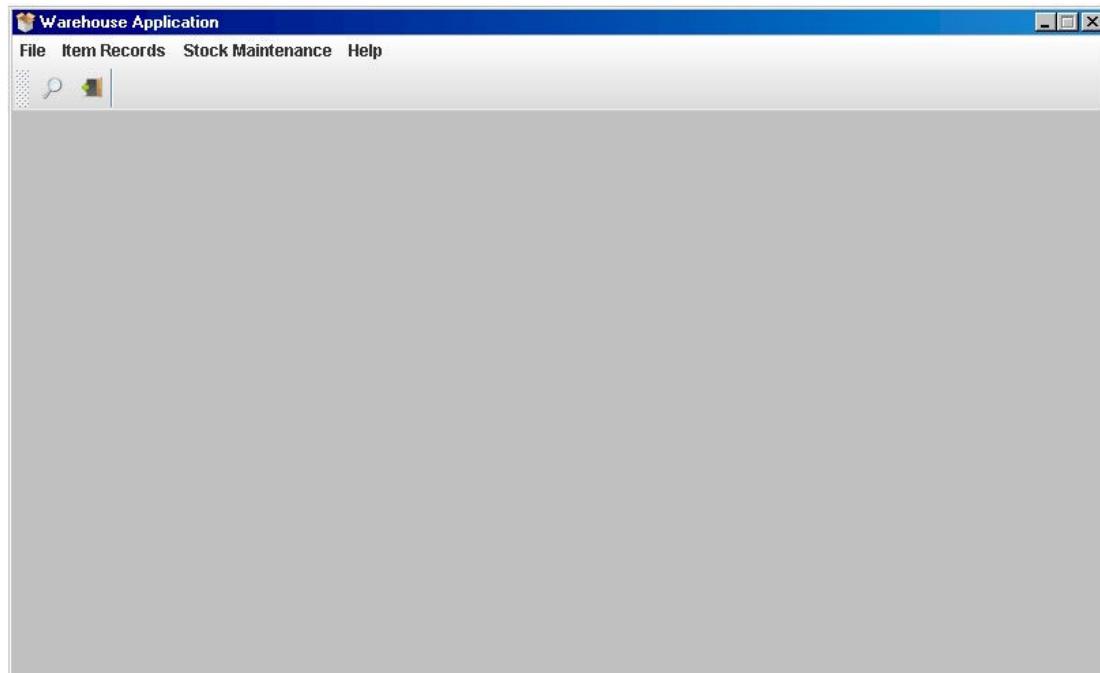
Optimal Pickup Route screen loads



The 'Finish' button is pressed

*Result*

Application returns to landing screen



A further pickup order will be provided to verify the optimal route algorithm. The same steps from above are repeated.

Three items are added to the item list and 'Next Step' is pressed. The appropriate quantities are entered. 'Next Step' is pressed.

The screenshot shows the 'Warehouse Application' interface. The top menu bar includes File, Item Records, Stock Maintenance, Admin, and Help. Below the menu is a toolbar with icons for file operations. The main area displays two screens sequentially:

**4. Item Exit**  
Fields marked with an \* are required

Item Exit Voucher No.*:	49328409	Dispatched By:	admin
Requested By:	S. Alvarez	Date:	25/7/2010

**5. Quantities**  
Fill the 'Quantity' column for each item to be removed. Make sure that this amount is not larger than the current stock or less than or equal to 0

Code	Name	U.M.	In Stock	Quantity*
490111	Orange alert vests	Unit (U)	4	1
301298	1/2" wide nylon rope	Metres...	10	6
709125	Steel Nails in 1/16" width	Thous...	2	2

**Next Step**

### Result

The user is informed that the transaction followed through.  
The Optimal Route screen loads with an appropriate pickup route.

The screenshot shows the 'Warehouse Application' interface. The top menu bar includes File, Item Records, Stock Maintenance, Admin, and Help. Below the menu is a toolbar with icons for file operations. The main area displays the 'Optimal Route' screen:

**6. Optimal Route**  
The column 'Order' denotes the pickup route

Order	Code	Name	UM	Location
1	490111	Orange alert vests	Unit (U)	A01.03.E05
2	709125	Steel Nails in 1/16" width	Thousands (MLL)	A01.28.A05
3	301298	1/2" wide nylon rope	Metres (M)	A01.40.C06

---

## D2 – Evaluating Solutions

---

The computer-based solution to address the problem of item entry to and withdrawal from the warehouse is an application that attempts to facilitate this process by implementing functionality not available in the application currently in place in Cementos Lima.

With regards to the proposed objectives that the application should fulfil, it was successful in addressing all objectives and mostly successful in satisfying them fully. With this application, a warehouse employee is able to speed up the process of item transactions by being able to search for them as a group, rather than being limited to individual item search. In addition, he/she can keep a detailed record of the items stored in the warehouse, and view the transactions on them over the years. Finally, full user functionality is implemented, with a system administrator having the ability to monitor and alter user information or permissions. All these functions have been tested for multiple data sets and have performed appropriately in all cases.

The design of the application was appropriate and was mostly carried out exactly in the building phase. A big strength that this application has is its background utilities, namely the abstract data structures in place. The binary trees for search were appropriate as they minimize the time needed to search for an item. Furthermore, an algorithm ensures that the tree is populated so that it will be a balanced tree. Though initially alternatives such as a linked list of items were considered, in the end I decided to implement a binary tree for its theoretical efficiency and capability for binary search. This proved true: the application's search function, the 'backbone' of the application, is efficient and fast.

Though successful as regards the proposed criteria for success, the application has some limitations when addressing the problem. The most important one is that in its current state, the application is stand-alone and is not integrated in any way with the warehouse application in place. If a user were to create an item, he/she would have to do so both in this application and in the existing application for accounting purposes. The lack of integration would effectively nullify the efficiency improvement that this application provides.

Furthermore, though this application provides the user a good route for item pickup, this route is not necessarily the optimal one. It relies on the general location of the item rather than the specific distance to and from aisles, which would help devise a more specific and less time consuming route. As the head of the warehouse repeatedly stressed, every second counts. Though the application is a better approach than no route, the optimality of the route is limited.

A better way to approach the solving of the optimal route problem would be using the graph abstract data structure. It allows one to 'plot' vertices away from the origin, at specified distances. These vertices would be the various locations to store items in the warehouse. There would be a certain 'cost' to travel between vertices. Using an algorithm called Djikstra's Algorithm, the most efficient route, i.e. the one that 'costs' less, can be calculated. This would provide a far more optimal item pickup route.

To solve the limitations and address the problem even better, some additional features could be implemented to the application. The application could be adapted so that it integrates with the current system, thus eliminating the need for having to carry out some tasks twice.

The user is not given the option to print out the route that the application provides. Though this functionality is not pivotal to the general process, it would certainly be a feature that might make the pickup process more efficient, especially with exit orders containing numerous items.

Some discussions are going on at Cementos Lima to integrate bar code technology into the warehouse. If implemented, this functionality would certainly be useful in processing item withdrawals. Rather than having to manually search for the items, a scanner would read a bar code, and the user would only have to enter the quantity of the item that is being withdrawn.

A component of the general warehouse process that the application does not address is item identification inside the warehouse. This feature is not directly relevant to the task at hand, and time constraints did not permit this feature to be developed. However, it would be logical and useful for a user creating an item to print out the item tag. Thus, all items would have a common identification system.

The initial design of the application was successful in terms of efficiency. Searching for an item in the application is smooth and with no major delays. This is due to the implementation of fully indexed files that populate a balanced binary tree, that result in a search with Big O efficiency of  $O(\log n)$ . Considering that the number of items in the warehouse may reach very high numbers, efficient search is the backbone of the application.

However, for the sake of data security, better memory handling and even faster search, a commercial SQL database could be implemented. This would eliminate the need of having to create and maintain index files and a background data structure running constantly, as the database would keep indexes and would run more efficient search algorithms. Furthermore, the data would be more protected as indexes would not have to be stored in text files.

The fact that the application uses the Swing library to render its components is a drain in efficiency, as it is a memory drain and makes the application slow at times. By writing the GUI components in a way where only the active screen consumes resources while all others remain inactive, this problem could be resolved.

The Items random access file takes up a lot of memory, as the Item Name and Description field require many bytes. This might be a problem as the number of items increases. Reducing the maximum number of characters of the Item Name or Description is not an option as the user specifically requested them to be a particular length. However, as the Description field is not always required and is in fact left blank many times, the item could have a flag indicating if it contains a description or not. Item Descriptions could be stored in another file, and they would be retrieved if the flag indicated that the item has a description.

To protect the integrity and security of the data in Transaction files, these could be written to a random access file instead of a text file that is easily read and modified. In a random access file, the data would not be directly readable and would be safer, a valuable asset given the importance of the data.