CIS 163 – Computer Science II

88 / 100

Project 3: "RedBox" Rental Program

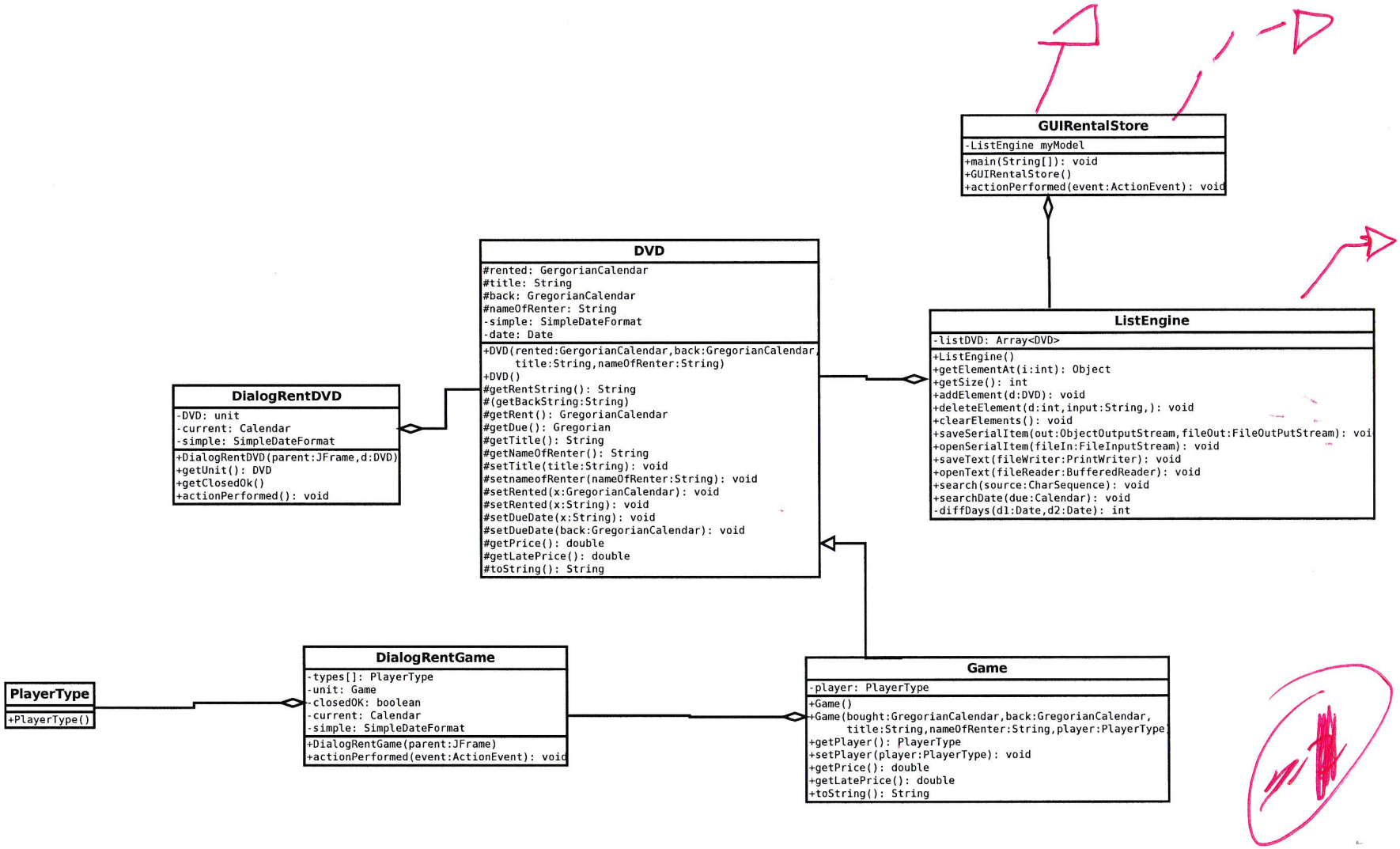
Student Name Aaron Teague	
Due Date Oct 29	
Date Submitted, Days Late, Late Penalty	

Graded Item	Points Assigned	Points Secured
Javadoc Comments and Coding Style/Technique (http://www.cis.gvsu.edu/studentsupport/javaguide) • Javadoc Comments and Coding Style/Technique • (http://www.cis.gvsu.edu/studentsupport/javaguide) • Code Indentation (auto format source code in IDE) • Naming Conventions (see Java style guide) • Proper access modifiers for fields and methods • Use of helper (private) methods • Using good variable names • Header/class comments • Every method uses @param and @return • Every method uses a /********** separator • Overall layout, readability, No text wrap • Using /** … / for each Instance variable • Has many inner "inner" comments	10	UML -1 -5 NO INITIAL UML
Steps 1 – 7: Basic Functionality • `DVD` and `Game` classes • `ListEngine` class • `GUIRentalStore` class • `DialogRentDVD` and `DialogRentGame` classes • Rent DVD (input via `DialogRentDVD`) • Rent Game (input via `DialogRentGame`) • Return DVD/Game • Menu item to save items in store as a serialized file • Menu item to load items from a serialized file • Final UML class diagram	65	
Step 8 – 9: Late Charge and Save/Load As Text • Menu item to save items in store as a text file • Menu item to load items from a text file • Due date field in return dialog box and late charge	10	
Step 10 – 12: Search and Sort Functionality • Menu item to search rental titles with a given string • Menu item to display items that are past due date • Items displayed in sorted order by title field	5 5 5	-1 NO -5
Total	**100**	

Comments:

All set

Have A set cost in DVD & GAME. Your was close. Spelling !!! SEE DVD


GUIRentalStore
-ListEngine myModel
+main(String[]): void
+GUIRentalStore()
+actionPerformed(event:ActionEvent): void
DVD
#rented: GergorianCalendar
#title: String
#back: GregorianCalendar
#nameOfRenter: String
-simple: SimpleDateFormat
-date: Date
+DVD(rented:GergorianCalendar,back:GregorianCalendar,
title:String,nameOfRenter:String)
+DVD()
#getRentString(): String
#(getBackString:String)
#getRent(): GregorianCalendar
#getDue(): Gregorian
#getTitle(): String
#getNameOfRenter(): String
#setTitle(title:String): void
#setnameofRenter(nameOfRenter:String): void
#setRented(x:GregorianCalendar): void
#setRented(x:String): void
#setDueDate(x:String): void
#setDueDate(back:GregorianCalendar): void
#getPrice(): double
#getLatePrice(): double
#toString(): String
ListEngine
-listDVD: Array<DVD>
+ListEngine()
+getElementAt(i:int): Object
+getSize(): int
+addElement(d:DVD): void
+deleteElement(d:int,input:String,): void
+clearElements(): void
+saveSerialItem(out:ObjectOutputStream,fileOut:FileOutPutStream): voi
+openSerialItem(fileIn:FileInputStream): void
+saveText(fileWriter:PrintWriter): void
+openText(fileReader:BufferedReader): void
+search(source:CharSequence): void
+searchDate(due:Calendar): void
-diffDays(d1:Date,d2:Date): int
DialogRentDVD
-DVD: unit
-current: Calendar
-simple: SimpleDateFormat
+DialogRentDVD(parent:JFrame,d:DVD)
+getUnit(): DVD
+getClosedOk()
+actionPerformed(): void
PlayerType
+PlayerType()
DialogRentGame
-types[]: PlayerType
-unit: Game
-closedOK: boolean
-current: Calendar
-simple: SimpleDateFormat
+DialogRentGame(parent:JFrame)
+actionPerformed(event:ActionEvent): void
Game
-player: PlayerType
+Game()
+Game(bought:GregorianCalendar,back:GregorianCalendar,
title:String,nameOfRenter:String,player:PlayerType)
+getPlayer(): PlayerType
+setPlayer(player:PlayerType): void
+getPrice(): double
+getLatePrice(): double
+toString(): String

```
 1 package package1;
 2
 3 import java.io.*;
10
11 public class ListEngine extends AbstractListModel {
12
13         /**ArracyList of DVD objects **/
14         private ArrayList<DVD> listDVD;
15
16         /*******************************************************
17          Default constructor creates initializes ArrayList
18          *******************************************************/
19         public ListEngine(){
20
21                 listDVD = new ArrayList();
22         }
23
24         /*************************************
25          Gets the object from the ArrayList
26          @param the index of the item
27          @return DVD object
28          *************************************/
29         @Override
30         public Object getElementAt(int i) {
31
32                 return listDVD.get(i);
33         }
34
35         /*************************************
36          Gets the object from the ArrayList
37          @param the index of the item
38          @return DVD object
39          *************************************/
40         @Override
41         public int getSize() {
42
43                 return listDVD.size();
44         }
45
46         /*************************************
47          Adds DVD object to the array.
48          @param the index of the item
49          *************************************/
50         public void addElement(DVD d){
51                 listDVD.add(d);
52                 fireIntervalAdded(this, 0, listDVD.size());
53
54         }
55
```

```
56          /*****************************************************
57           Removes the object from the array and update the
58           JList. And calculates the rate for the rental.
59           @param the index of the item and the date of the return.
60           *****************************************************/
61          public void deleteElement(int d, String input){
62
63                  Calendar x = new GregorianCalendar();
64                  SimpleDateFormat simple = new SimpleDateFormat("MM/dd/
    yyyy");
65                  double priceOri = listDVD.get(d).getPirce();;
66                  double total = 0.0;
67                  double original = 0.0;
68                  double priceLate = listDVD.get(d).getLatePirce();
69                  NumberFormat formatter = NumberFormat.getCurrencyInstance();
70
71
72
73                  try {
74                          Date date = simple.parse(input);
75                          x.setTime(date);
76                  } catch (ParseException e) {
77                   JOptionPane.showMessageDialog(null, "Sorry we'll we need at
    "+
78                                                  "date please try again.");
79                  }
80
81                  //Check to make sure the entered date is after the rent date
82          if(listDVD.get(d).getRent().before(x)){
83
84                  //Set the DVD values to variables
85                  String name = listDVD.get(d).getNameOfRenter();
86                  String title = listDVD.get(d).getTitle();
87                  GregorianCalendar due = listDVD.get(d).getDue();
88                  GregorianCalendar rent = listDVD.get(d).getRent();
89
90                  //Checks if the return date is late
91                  if(x.before(due) || x.equals(due)){
92                  total = (diffDays(x.getTime(),
93                  listDVD.get(d).getRent().getTime())) * priceOri;
94
95                  }
96
97                  else
98                  {
99                   //Caluclates the price before late charge
100                  original = (diffDays(due.getTime(),rent.getTime()))
101                                  * priceOri;
102                  //Calculates and adds late charge
```

```
103                 total = original + diffDays(x.getTime(), due.getTime()) *
104                                 priceLate;
105                 }
106
107                 JOptionPane.showMessageDialog(null,"Thank you " + name +
108                                             " for renting " + title + " you owe
    " +
109                                             formatter.format(total));
110                 listDVD.remove(d);
111                 fireContentsChanged(this, 0, listDVD.size());
112                 }
113
114             else
115             {
116
117                JOptionPane.showMessageDialog(null,"Please try a " +
118                                                 "different date.");
119             }
120
121
122         }
123
124         /*******************************************************
125          Removes all objects from the array and update the
126          JList.
127          *******************************************************/
128         public void clearElements(){
129                 this.listDVD.clear();
130                 fireIntervalRemoved(this, 0, listDVD.size());
131
132         }
133
134
135         /*******************************************************
136          Save objects to a Serial file
137          *******************************************************/
138         public void saveSerialItem(ObjectOutputStream out,
139                         FileOutputStream fileOut) throws IOException{
140
141                 for(int i = 0; i < listDVD.size(); i++){
142                         out.writeObject(listDVD.get(i));
143                 }
144
145                 out.close();
146                 fileOut.close();
147
148         }
149
150         /*******************************************************
```

```
151         Loads the array from a serial file.
152         @param fileIn
153         ****************************************************/
154        public void openSerialItem(FileInputStream fileIn) throws
155        IOException, ClassNotFoundException{
156
157                InputStream buffer = new BufferedInputStream(fileIn);
158                try
159                {
160                ObjectInputStream in = new ObjectInputStream(buffer);
161
162                //Clears Jlist to show new database
163                this.clearElements();
164
165                while(in != null){
166                        try{
167                                DVD dvd = null;
168
169                                dvd = (DVD) in.readObject();
170                                this.addElement(dvd);
171
172                        }
173                        catch(EOFException x){
174                                break;
175                        }
176                }
177
178                in.close();
179                fileIn.close();
180               }
181               catch(StreamCorruptedException e) {
182
183                   JOptionPane.showMessageDialog(null,"Please try a "+
184                           "different file.");
185               }
186
187        }
188
189        /*****************************************************
190         Save objects to a text file
191         @param filewriter, text file
192         *****************************************************/
193        public void saveText(PrintWriter filewriter){
194        //Prints the object info: "Name, Date..."
195                String line = " ";
196                String lineTwo = " ";
197                for(int i = 0; i < listDVD.size(); i++){
198                        DVD dvdUnit = listDVD.get(i);
199                        line = dvdUnit.getClass().getName() + "," +
```

```
200                         dvdUnit.getNameOfRenter() + "," + dvdUnit.getTitle()
   + "," +
201                         dvdUnit.getRentString() + "," +
   dvdUnit.getBackString();
202
203                         if(dvdUnit instanceof Game){
204                                 lineTwo = line + "," +
   ((Game)dvdUnit).getPlayer();
205                                 filewriter.println(lineTwo);
206                         }
207
208                         else{
209
210                                 filewriter.println(line);
211                         }
212
213                 }
214
215                 filewriter.close();
216
217
218         }
219
220         /*******************************************************
221          Loads objects to the array List from a text file
222          @param filereader, text file
223          *******************************************************/
224         public void openText(BufferedReader filereader)
225                         throws IOException, ParseException{
226
227                 this.clearElements();
228
229                 String info = filereader.readLine();
230
231                 while(info != null){
232                         //Splits the elements in text to strings
233                         String[] listInfo = info.split(",");
234
235                         //Checks if the object is a DVD or Game
236                         if(listInfo[0].equals("package1.DVD"))
237                         {
238                                 DVD newDVD = new DVD();
239                                 newDVD.setnameOfRenter(listInfo[1]);
240                                 newDVD.setTitle(listInfo[2]);
241                                 newDVD.setRented(listInfo[3]);
242                                 newDVD.setDueDate(listInfo[4]);
243                                 this.addElement(newDVD);
244                         }
245
```

```
246
247                         else
248                         {
249                                 Game newGame = new Game();
250                                 newGame.setnameOfRenter(listInfo[1]);
251                                 newGame.setTitle(listInfo[2]);
252                                 newGame.setRented(listInfo[3]);
253                                 newGame.setDueDate(listInfo[4]);
254                                 newGame.setPlayer(PlayerType.valueOf
    (listInfo[5]));
255                                 this.addElement(newGame);
256                         }
257
258                         info = filereader.readLine();
259                 }
260
261         }
262         /*******************************************************
263          Searchs through the Array list to find all objects
264          contain the CharSequence
265          @param the sequence your searching for
266          *******************************************************/
267         public void search(CharSequence source){
268                 try{
269
270                         ArrayList<DVD> match = new ArrayList();
271
272                         for(int i = 0; i < listDVD.size(); i++){
273                                 //Fills an array list with matches
274                                 if(listDVD.get(i).getTitle().contains
    (source))
275                                         match.add(listDVD.get(i));
276                         }
277
278                         //Clears the elements
279                         this.clearElements();
280
281                         //Adds the elements found
282                         for(int i = 0; i < match.size(); i++){
283
284                                 this.addElement(match.get(i));
285                         }
286                 }
287
288                 catch(NullPointerException e){
289
290                 }
291
292
```

```
293         }
294         /*******************************************************
295         Finds rentals past due.
296         @param Due date
297         *******************************************************/
298        public void searchDate(Calendar due){
299             String output = "";
300             for(int i = 0; i < listDVD.size(); i++){
301                 if(listDVD.get(i).getDue().getTime().after(due.getTime())){
302                 int days = diffDays(listDVD.get(i).getDue().getTime(),
    due.getTime());
303                 output += listDVD.get(i).toString() +" "  + days + " days
    \n";
304                 }
305             }
306             JOptionPane.showMessageDialog(null, output);
307
308         }
309
310         /*******************************************************
311          Calculates the number day between two dates.
312          I found this method on stackoverflow.com
313          @param Date d1, Date d2
314          @return the difference between d1 and d2
315          *******************************************************/
316         private int diffDays(Date d1, Date d2){
317
318                 return (int)((d1.getTime() - d2.getTime()) /
319                                 (1000 * 60 * 60 * 24));
320
321         }
322
323
324 }
```

```
 1 package package1;
 2
 3 import java.awt.*;
11
12
13 public class GUIRentalStore extends JFrame implements ActionListener{
14
15         private static final long serialVersionUID = 1L;
16
17         JMenuBar menus;
18         JMenu fileMenu;
19         JMenuItem exitItem;
20         JMenuItem openSerialItem;
21         JMenuItem saveSerialItem;
22         JMenuItem openTextItem;
23         JMenuItem saveTextItem;
24         JMenu actionMenu;
25         JMenuItem rentDVDItem;
26         JMenuItem rentGameItem;
27         JMenuItem returnItem;
28         JMenuItem dueDateItem;
29         JMenuItem searchTitleItem;
30         JList myList;
31         JFileChooser chooser;
32         JScrollPane myScroll;
33         ListEngine myModel;
34
35
36         public static void main(String[] args){
37                 new GUIRentalStore();
38         }
39
40         public GUIRentalStore(){
41                 menus = new JMenuBar();
42                 fileMenu = new JMenu ("File");
43                 exitItem = new JMenuItem("Exit!");
44                 openSerialItem = new JMenuItem("Open Serial...");
45                 saveSerialItem = new JMenuItem("Save Serial...");
46                 openTextItem = new JMenuItem("Open Text...");
47                 saveTextItem = new JMenuItem("Save Text...");
48
49                 fileMenu.add(openSerialItem);
50                 fileMenu.add(saveSerialItem);
51                 fileMenu.add(openTextItem);
52                 fileMenu.add(saveTextItem);
53                 fileMenu.add(exitItem);
54                 menus.add(fileMenu);
55
56                 openSerialItem.addActionListener(this);
```

```
57                saveSerialItem.addActionListener(this);
58                openTextItem.addActionListener(this);
59                saveTextItem.addActionListener(this);
60                exitItem.addActionListener(this);
61
62
63                actionMenu = new JMenu ("Action");
64                rentDVDItem = new JMenuItem("Rent DVD");
65                rentGameItem = new JMenuItem("Rent Game");
66                returnItem = new JMenuItem("Return");
67                searchTitleItem = new JMenuItem("Search Title");
68                dueDateItem = new JMenuItem("Search by Due Date");
69
70                actionMenu.add(rentDVDItem);
71                actionMenu.add(rentGameItem);
72                actionMenu.add(returnItem);
73                actionMenu.add(searchTitleItem);
74                actionMenu.add(dueDateItem);
75                menus.add(actionMenu);
76
77                rentDVDItem.addActionListener(this);
78                rentGameItem.addActionListener(this);
79                returnItem.addActionListener(this);
80                searchTitleItem.addActionListener(this);
81                dueDateItem.addActionListener(this);
82
83
84                myModel = new ListEngine();
85                myList = new JList(myModel);
86                myScroll = new JScrollPane(myList);
87                chooser = new JFileChooser();
88
89                this.setJMenuBar(menus);
90                this.add(myScroll);
91
92
93                this.setSize(700, 700);
94                this.setVisible(true);
95
96        }
97
98        public void actionPerformed(ActionEvent event) {
99                if(event.getSource() == exitItem)
100                       System.exit(1);
101
102               if(event.getSource() == openSerialItem){
103                       chooser.showOpenDialog(null);
104                       try {
105                               FileInputStream fileIn =
```

```
106                                 new FileInputStream(chooser.getSelectedFile
   ());
107                                 myModel.openSerialItem(fileIn);
108                         } catch (FileNotFoundException e) {
109                                 // TODO Auto-generated catch block
110                                 e.printStackTrace();
111                         } catch (IOException e) {
112                                 // TODO Auto-generated catch block
113                                 e.printStackTrace();
114                         } catch (ClassNotFoundException e) {
115                                 // TODO Auto-generated catch block
116                                 e.printStackTrace();
117                         } catch (NullPointerException e) {
118                                 // TODO Auto-generated catch block
119
120                         }
121
122                 }
123
124
125                 if(event.getSource() == saveSerialItem){
126                         chooser.showSaveDialog(null);
127                         try {
128                                 FileOutputStream fileOut =
129                                 new FileOutputStream(chooser.getSelectedFile
   ());
130                                 ObjectOutputStream out = new
   ObjectOutputStream(fileOut);
131                                 myModel.saveSerialItem(out, fileOut);
132                         } catch (FileNotFoundException e) {
133                                 // TODO Auto-generated catch block
134                                 e.printStackTrace();
135                         } catch (IOException e) {
136                                 // TODO Auto-generated catch block
137                                 e.printStackTrace();
138                         } catch (NullPointerException e) {
139                                 // TODO Auto-generated catch block
140
141                         }
142                 }
143
144
145                 if(event.getSource() == openTextItem){
146                         chooser.showOpenDialog(null);
147                         File file = chooser.getSelectedFile();
148                         try {
149                                 BufferedReader getInfo = new BufferedReader
   (new FileReader(file));
150                                 myModel.openText(getInfo);
```

```
151
152                         } catch (FileNotFoundException e) {
153                                 // TODO Auto-generated catch block
154                                 e.printStackTrace();
155                         } catch (IOException e) {
156                                 // TODO Auto-generated catch block
157                                 e.printStackTrace();
158                         } catch (ParseException e) {
159                                 // TODO Auto-generated catch block
160                                 e.printStackTrace();
161                         } catch (NullPointerException e) {
162                                 // TODO Auto-generated catch block
163
164                         } catch (ArrayIndexOutOfBoundsException e){
165                             JOptionPane.showMessageDialog(null, "Check
    file.");
166                         }
167                 }
168
169
170                 if(event.getSource() == saveTextItem){
171                         chooser.showSaveDialog(null);
172                 File file = chooser.getSelectedFile();
173                 try {
174                         PrintWriter filewriter = new PrintWriter(new
    BufferedWriter(new FileWriter(file)));
175                         myModel.saveText(filewriter);
176                 } catch (IOException e) {
177                         // TODO Auto-generated catch block
178                         e.printStackTrace();
179                 } catch (NullPointerException e) {
180                         // TODO Auto-generated catch block
181
182                 }
183
184         }
185
186
187                 if(event.getSource() == rentDVDItem) {
188                         DVD d = new DVD();
189                         DialogRentDVD dialog = new DialogRentDVD(this, d);
190                         if(dialog.getClosedOk())
191                         {
192                                 myModel.addElement(dialog.getUnit());
193                         }
194
195                 }
196
197                 if(event.getSource() == rentGameItem) {
```

```
198                         Game game = new Game();
199                         DialogRentGame dialog = new DialogRentGame(this,
    game);
200                                 if(dialog.getClosedOk())
201                                         myModel.addElement(dialog.getUnit
    ());
202
203                 }
204
205
206                 if(event.getSource() == returnItem){
207                         int selectedIndex = myList.getSelectedIndex();
208                         try{
209                         String source = JOptionPane.showInputDialog(null,
    "Please enter a date.");
210
211                         if(!source.isEmpty()){
212                         if(selectedIndex != -1)
213                                 myModel.deleteElement(selectedIndex,
    source);
214                         }
215                         else
216                         {
217                                 JOptionPane.showMessageDialog(null, "Sorry
    we need a date to " +
218                         "complete search.");
219                         }
220                         }
221                         catch(NullPointerException e){
222
223                         }
224                 }
225
226                 if(event.getSource() == searchTitleItem){
227                         CharSequence source = " ";
228                         //JFrame input = new JFrame();
229                         source = JOptionPane.showInputDialog("Search for
    for?");
230                         myModel.search(source);
231                 }
232
233                 if(event.getSource() == dueDateItem){
234                         String input =
235                         JOptionPane.showInputDialog("Search for what
    date?");
236                         SimpleDateFormat simple = new SimpleDateFormat("MM/
    dd/yyyy");
237                         Calendar x = new GregorianCalendar();
238                         try {
```

```
239                             Date date = simple.parse(input);
240                             x.setTime(date);
241                         } catch (ParseException  e) {
242                             // TODO Auto-generated catch block
243                             e.printStackTrace();
244                         }
245                         catch(NullPointerException e){
246
247                         }
248
249                         myModel.searchDate(x);
250
251
252                 }
253
254         }
255
256
257
258
259 }
```

```java
package package1;


import java.awt.*;

public class DialogRentDVD extends JDialog implements ActionListener{

    private static final long serialVersionUID = 1L;
    /** Renter's name label**/
    private JLabel nameLb;
    /** The tile label**/
    private JLabel titleLb;
    /** Date rented on label**/
    private JLabel rentDateLb;
    /** Due date label**/
    private JLabel dueDateLb;
    /** Renter's name text**/
    private JTextField nameTxt;
    /** The tile text**/
    private JTextField titleTxt;
    /** Date rented on text**/
    private JTextField rentDateTxt;
    /** Due date label**/
    private JTextField dueDateTxt;
    /** Cancel button**/
    private JButton cancelButton;
    /** Ok button**/
    private JButton okButton;
    /** Panel display rental info**/
    private JPanel panelOne;
    /** Panel display buttons**/
    private JPanel panelTwo;
    /** DVD object**/
    private DVD unit;
    /** Knows if all info was collected correctly**/
    private boolean closedOk;
    /** The current date**/
    private Calendar current;
    /** Date format**/
    private SimpleDateFormat simple;

    /*************************************************
     * Creates the DVD Dialog Box
     * @param JFrame and DVD object
     * @return the DVD dialog box
     *************************************************/
    public DialogRentDVD(JFrame parent, DVD d)
    {
        super(parent, true);
        closedOk = false;
        unit = d;
        simple = new SimpleDateFormat("MM/dd/yyyy");
        current = Calendar.getInstance();
        cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(this);
```

```
63          okButton  = new JButton("Ok");
64          okButton.addActionListener(this);
65          nameLb = new JLabel("Your Name");
66          titleLb = new JLabel("Title");
67          rentDateLb = new JLabel("Rented on Date");
68          dueDateLb = new JLabel("Due Back");
69          nameTxt = new JTextField();
70          titleTxt = new JTextField();
71          rentDateTxt = new JTextField(simple.format(current.getTime()));
72          current.add(Calendar.DATE, 1);
73          dueDateTxt = new JTextField(simple.format(current.getTime()));
74          GridLayout newLayout = new GridLayout(6,2);
75
76          panelOne = new JPanel();
77          panelOne.setLayout(newLayout);
78          panelOne.add(nameLb);
79          panelOne.add(nameTxt);
80          panelOne.add(titleLb);
81          panelOne.add(titleTxt);
82          panelOne.add(rentDateLb);
83          panelOne.add(rentDateTxt);
84          panelOne.add(dueDateLb);
85          panelOne.add(dueDateTxt);
86
87          panelTwo =  new JPanel();
88          panelTwo.add(cancelButton);
89          panelTwo.add(okButton);
90
91          getContentPane().add(panelOne, BorderLayout.NORTH);
92          getContentPane().add(panelTwo, BorderLayout.CENTER);
93
94          setSize(300, 300);
95          setVisible(true);
96
97
98      }
99
100     /*****************************************************
101      Returns the DVD unit
102      @param None
103      @return The DVD
104      *****************************************************/
105     public DVD getUnit(){
106         return unit;
107     }
108
109     /*****************************************************
110      Returns boolean value true if all info was collected
111      @param None
112      @return true value if all info was collected
113      *****************************************************/
114     public boolean getClosedOk(){
115         return closedOk;
116     }
117
```

```
118     /*****************************************************
119      Action listener
120      @param Action event
121      *****************************************************/
122     @Override
123     public void actionPerformed(ActionEvent event) {
124
125         if(event.getSource() == cancelButton){
126         //Does nothing and shuts off dialog box if the cancel
127         //button is selected
128           setVisible(false);
129           dispose();
130         }
131
132         if(event.getSource() == okButton){
133             closedOk = true;
134
135             Calendar x = new GregorianCalendar();
136
137             //Checks to make sure if the date is valid
138             try {
139                 Date date = simple.parse(rentDateTxt.getText());
140                 x.setTime(date);
141             } catch (ParseException e) {
142             JOptionPane.showMessageDialog(null,"Please enter a cor" +
143                         "rect rent date.");
144                 closedOk = false;
145                 setVisible(false);
146                 dispose();
147             }
148
149             //sets rented date
150             unit.setRented((GregorianCalendar)x);
151
152
153             Calendar a = new GregorianCalendar();
154             //Checks if date is valid
155             try {
156                 Date dateTwo = simple.parse(dueDateTxt.getText());
157                 a.setTime(dateTwo);
158             } catch (ParseException e) {
159             JOptionPane.showMessageDialog(null,"Please enter a cor" +
160                         "rect due date.");
161                 closedOk = false;
162                 setVisible(false);
163                 dispose();
164             }
165
166             //sets due date
167             unit.setDueDate((GregorianCalendar)a);
168
169
170             unit.setTitle(titleTxt.getText());
171             unit.setnameOfRenter(nameTxt.getText());
172
```

```
173             //Checks if the title is valid
174             if(unit.getTitle().isEmpty()){
175              JOptionPane.showMessageDialog(null,"Please enter a cor" +
176                          "rect Title.");
177                 closedOk = false;
178                 setVisible(false);
179                 dispose();
180 
181             }
182 
183             //Checks if the name is valid
184             if(unit.getNameOfRenter().isEmpty()){
185              JOptionPane.showMessageDialog(null,"Please enter a cor" +
186                          "rect Name.");
187                 closedOk = false;
188                 setVisible(false);
189                 dispose();
190 
191             }
192 
193             setVisible(false);
194             dispose();
195 
196 
197         }
198     }
199 
200 }
201 
```

```
 1 package package1;
 2 import java.awt.*;
12
13 public class DialogRentGame extends JDialog implements ActionListener{
14
15     private static final long serialVersionUID = 1L;
16     /**Name label **/
17     private JLabel nameLb;
18     /**Title label **/
19     private JLabel titleLb;
20     /**Rent date label **/
21     private JLabel rentDateLb;
22     /**Due label **/
23     private JLabel dueDateLb;
24     /**Playertype label **/
25     private JLabel playerLb;
26     /**Name text **/
27     private JTextField nameTxt;
28     /**Title text **/
29     private JTextField titleTxt;
30     /**Rent date text **/
31     private JTextField rentDateTxt;
32     /**Due text **/
33     private JTextField dueDateTxt;
34     /**Cancel button **/
35     private JButton cancelButton;
36     /**Ok button **/
37     private JButton okButton;
38     /**Combobox holds the playertye **/
39     private JComboBox playerCombo;
40     /**Panel holds the info **/
41     private JPanel panelOne;
42     /**Panel holds the buttons **/
43     private JPanel panelTwo;
44     /**Array holds the playertype values **/
45     private PlayerType types[];
46     /**Game object **/
47     private Game unit;
48     /** Knows if all info was collected correctly**/
49     private boolean closedOk;
50     /** The current date**/
51     private Calendar current;
52     /** Formats the date**/
53     private SimpleDateFormat simple;
54
55     /*****************************************************
56      * Creates the the Game Dailog Box
57      * @param The frame and game object
58      * @return The game dialog box
59      *****************************************************/
60     public DialogRentGame(JFrame parent, Game d)
61     {
62         super(parent, true);
63         closedOk = false;
64         current = Calendar.getInstance();
```

```
65          simple = new SimpleDateFormat("MM/dd/yyyy");
66          unit = d;
67          types = new PlayerType[4];
68          types[0] = PlayerType.XBox360;
69          types[1] = PlayerType.PS3;
70          types[2] = PlayerType.PS4;
71          types[3] = PlayerType.XBoxOne;
72          cancelButton = new JButton("Cancel");
73          cancelButton.addActionListener(this);
74          okButton  = new JButton("Ok");
75          okButton.addActionListener(this);
76          nameLb = new JLabel("Your Name");
77          titleLb = new JLabel("Title");
78          rentDateLb = new JLabel("Rented on Date");
79          dueDateLb = new JLabel("Due Back");
80          playerLb = new JLabel("Type of Player");
81          playerCombo = new JComboBox(types);
82          nameTxt = new JTextField();
83          titleTxt = new JTextField();
84          rentDateTxt = new JTextField(simple.format(current.getTime()));
85          current.add(Calendar.DATE, 1);
86          dueDateTxt = new JTextField(simple.format(current.getTime()));
87          GridLayout newLayout = new GridLayout(8,2);
88
89          panelOne = new JPanel();
90          panelOne.setLayout(newLayout);
91          panelOne.add(nameLb);
92          panelOne.add(nameTxt);
93          panelOne.add(titleLb);
94          panelOne.add(titleTxt);
95          panelOne.add(rentDateLb);
96          panelOne.add(rentDateTxt);
97          panelOne.add(dueDateLb);
98          panelOne.add(dueDateTxt);
99          panelOne.add(playerLb);
100         panelOne.add(playerCombo);
101
102
103         panelTwo =  new JPanel();
104         panelTwo.add(cancelButton);
105         panelTwo.add(okButton);
106
107
108         getContentPane().add(panelOne, BorderLayout.NORTH);
109         getContentPane().add(panelTwo, BorderLayout.CENTER);
110
111         setSize(300, 300);
112         setVisible(true);
113
114
115     }
116
117     public DVD getUnit(){
118         return unit;
119     }
```

```
120
121     public boolean getClosedOk(){
122         return closedOk;
123     }
124
125     /*********************************************
126      * Action even handler
127      * @param The event that happened
128      * @return None
129      *********************************************/
130     @Override
131     public void actionPerformed(ActionEvent event) {
132         //
133         if(event.getSource() == okButton){
134
135             closedOk = true;
136
137
138             Calendar x = new GregorianCalendar();
139
140             //sets the text to a date
141             try {
142                 Date date = simple.parse(rentDateTxt.getText());
143                 x.setTime(date);
144             } catch (ParseException e) {
145             JOptionPane.showMessageDialog(null,"Please enter a cor" +
146                         "rect rent date.");
147                 closedOk = false;
148                 setVisible(false);
149                 dispose();
150             }
151             //set the rented date
152             unit.setRented((GregorianCalendar)x);
153
154
155             Calendar a = new GregorianCalendar();
156
157             //sets the text to date
158             try {
159                 Date dateTwo = simple.parse(dueDateTxt.getText());
160                 a.setTime(dateTwo);
161             } catch (ParseException e) {
162             JOptionPane.showMessageDialog(null,"Please enter a cor" +
163                         "rect due date.");
164                 closedOk = false;
165                 setVisible(false);
166                 dispose();
167             }
168             //sets the text to a date
169             unit.setDueDate((GregorianCalendar)a);
170
171
172             unit.setPlayer(types[playerCombo.getSelectedIndex()]);
173
174
```

```
175             unit.setTitle(titleTxt.getText());
176             unit.setnameOfRenter(nameTxt.getText());
177 
178             //checks if the title is empty
179             if(unit.getTitle().isEmpty()){
180             JOptionPane.showMessageDialog(null,"Please enter a cor" +
181                             "rect Title.");
182                     closedOk = false;
183                     setVisible(false);
184                     dispose();
185 
186                 }
187 
188             //checks if the name is empty
189             if(unit.getNameOfRenter().isEmpty()){
190             JOptionPane.showMessageDialog(null,"Please enter a cor" +
191                             "rect Name.");
192                     closedOk = false;
193                     setVisible(false);
194                     dispose();
195 
196                 }
197 
198 
199             setVisible(false);
200             dispose();
201 
202 
203         }
204 
205         if(event.getSource() == cancelButton){
206               setVisible(false);
207               dispose();
208             }
209 
210     }
211 
212 }
213 
```

```java
package package1;

import java.text.SimpleDateFormat;

public class Game extends DVD{

	private static final long serialVersionUID = 1L;
	/**The player type **/
	private PlayerType player;

	/***********************************************
	 * Default constructor creates a blank game
	 **********************************************/
	public Game(){
		super();

	}

	/***********************************************
	 * Constructor with parameters
	 * @param bought
	 * @param back
	 * @param title
	 * @param nameOfRenter
	 * @param player
	 **********************************************/
	public Game(GregorianCalendar bought, GregorianCalendar back,
			String title, String nameOfRenter, PlayerType player){
		super(bought, back, title, nameOfRenter);
		this.player = player;

	}

	/***********************************************
	 * Gets the player type
	 * @return The Player type
	 **********************************************/
	public PlayerType getPlayer(){
		return player;
	}

	/***********************************************
	 * Sets the player type
	 * @param The Player type
	 **********************************************/
	public void setPlayer(PlayerType player){
		this.player = player;
	}

	/***********************************************
	 * Gets the standard price
	 * @return the standard price
	 **********************************************/
	public double getPirce(){
		return 5;
```

```
58          }
59
60      /*********************************************
61       *Gets the late price
62       *@return the late price
63       *********************************************/
64      public double getLatePirce(){
65          return 10;
66      }
67
68      /*********************************************
69       *Prints the game in a nice format
70       *@return The game facts
71       *********************************************/
72      public String toString(){
73
74          Calendar calRent = new GregorianCalendar();
75          Calendar calBack = new GregorianCalendar();
76          SimpleDateFormat simple = new SimpleDateFormat("MM/dd/yyyy");
77          calRent = rented;
78          calBack = back;
79
80
81
82              return  "Name: " + nameOfRenter + ",   Title: " + title +
83              ",   Rented on: " +
84              simple.format(calRent.getTime()) +
85              ",   Due back on: " + simple.format(calBack.getTime()) +
86              ",   Game Console: " + player.name();
87
88
89          }
90
91
92  }
93
```

```java
package package1;

import java.io.Serializable;

public class DVD implements Serializable{

    private static final long serialVersionUID = 1L;
    /**The date the DVD was rented **/
    protected GregorianCalendar rented;
    /**The date the DVD is back */
    protected GregorianCalendar back;
    /**The title of the DVD */
    protected String title;
    /**The name of the person who is renting the DVD */
    protected String nameOfRenter;
    /**Calendar varliable that helps print the date */
    private Calendar cal = new GregorianCalendar();
    /**Formats print the date */
    private SimpleDateFormat simple =
    new SimpleDateFormat("MM/dd/yyyy");
    /**Date variable that helps print the date */
    private Date date;

    /************************************************
     * Constructor creates the DVD object
     * @param rented
     * @param back
     * @param title
     * @param nameOfRenter
     ************************************************/
    public DVD(GregorianCalendar rented, GregorianCalendar back,
     String title, String nameOfRenter){
        this.back = back;
        this.rented = rented;
        this.title = title;
        this.nameOfRenter = nameOfRenter;
    }

    /****************************************
     * Constructor creates a new blank DVD object
     ****************************************/
    public DVD(){
        rented = new GregorianCalendar();
        back =  new GregorianCalendar();
        title = " ";
        nameOfRenter = " ";

    }

    /*************************************
     * Gets the rent date in a string
     * @return Rent date as a string
     *************************************/
    protected  String getRentString(){
        cal = rented;
```

```
61          return simple.format(cal.getTime());
62
63      }
64
65      /**************************************
66       * Gets the due date in a string
67       * @return Due date as a string
68       **************************************/
69      protected  String getBackString(){
70          cal = back;
71          return simple.format(cal.getTime());
72
73      }
74
75      /**********************************************
76       * Gets the Rent date as a GregorianCalendar
77       * @return Rent date as a GregorianCalendar
78       **********************************************/
79      protected GregorianCalendar getRent(){
80          return rented;
81      }
82
83      /**********************************************
84       * Gets the Due date as a GregorianCalendar
85       * @return Due date as a GregorianCalendar
86       **********************************************/
87      protected GregorianCalendar getDue(){
88
89          return back;
90      }
91
92      /**********************************************
93       * Gets the Title.
94       * @return The Title.
95       **********************************************/
96      protected String getTitle(){
97
98          return title;
99
100     }
101
102     /**********************************************
103      * Gets the name of renter.
104      * @return The name of renter.
105      **********************************************/
106     protected String getNameOfRenter(){
107
108         return nameOfRenter;
109
110     }
111
112     /**********************************************
113      * Sets the name of renter.
114      * @param The Title.
115      **********************************************/
```

```
116     protected void setTitle(String title){
117
118         this.title = title;
119     }
120
121     /*********************************************
122      * Sets the name of renter.
123      * @param The name of renter.
124      *********************************************/
125     protected void setnameOfRenter(String nameOfRenter){
126
127         this.nameOfRenter = nameOfRenter;
128     }
129
130     /*********************************************
131      * Sets the rented date.
132      * @param The rented date.
133      *********************************************/
134     protected void setRented(GregorianCalendar x){
135
136         this.rented = x;
137     }
138
139     /*********************************************
140      * Sets the rent date by String
141      * @param The rent date by String
142      *********************************************/
143     protected void setRented(String x) throws ParseException{
144         date = simple.parse(x);
145         rented.setTime(date);
146     }
147
148     /*********************************************
149      * Sets the due date by String
150      * @param due date by String
151      *********************************************/
152     protected void setDueDate(String x) throws ParseException{
153         date = simple.parse(x);
154         rented.setTime(date);
155     }
156
157     /*********************************************
158      * Sets the due date by String
159      * @param due date by String
160      *********************************************/
161     protected void setDueDate(GregorianCalendar back){
162
163         this.back = back;
164     }
165
166     /*********************************************
167      * Gets the standard price
168      * @param none
169      *********************************************/
170     protected double getPirce(){
```

```
171         return 1.2;
172     }
173 
174     /*********************************************
175      * Gets the late price
176      * @param none
177      *********************************************/
178     protected double getLatePirce(){
179         return 2;
180     }
181 
182     /*********************************************
183      * Prints the DVD in a nice format
184      * @param none
185      *********************************************/
186     public String toString(){
187         Calendar calBack = new GregorianCalendar();
188         SimpleDateFormat simple = new SimpleDateFormat("MM/dd/yyyy");
189         calBack = back;
190         return  "Name: " + nameOfRenter + ",   Title: " + title +
191                 ",   Rented on: " + this.getRentString() +
192                 ",   Due back on: " + this.getBackString();
193 
194     }
195 
196 }
197 
```

PlayerType.java

```
1 package package1;
2 /**Enum type for video game systems **/
3 public enum PlayerType {
4
5     XBox360,PS3,XBoxOne,PS4
6     }
7
```

```
 1 package package1;
 2
 3 import java.awt.*;
11
12
13 public class GUIRentalStore extends JFrame implements ActionListener{
14
15     private static final long serialVersionUID = 1L;
16
17     private JMenuBar menus;
18     private JMenu fileMenu;
19     private JMenuItem exitItem;
20     private JMenuItem openSerialItem;
21     private JMenuItem saveSerialItem;
22     private JMenuItem openTextItem;
23     private JMenuItem saveTextItem;
24     private JMenu actionMenu;
25     private JMenuItem rentDVDItem;
26     private JMenuItem rentGameItem;
27     private JMenuItem returnItem;
28     private JMenuItem dueDateItem;
29     private JMenuItem searchTitleItem;
30     private JList myList;
31     private JFileChooser chooser;
32     private JScrollPane myScroll;
33     private ListEngine myModel;
34
35
36     public static void main(String[] args){
37         new GUIRentalStore();
38     }
39
40     public GUIRentalStore(){
41         menus = new JMenuBar();
42         fileMenu = new JMenu ("File");
43         exitItem = new JMenuItem("Exit!");
44         openSerialItem = new JMenuItem("Open Serial...");
45         saveSerialItem = new JMenuItem("Save Serial...");
46         openTextItem = new JMenuItem("Open Text...");
47         saveTextItem = new JMenuItem("Save Text...");
48
49         fileMenu.add(openSerialItem);
50         fileMenu.add(saveSerialItem);
51         fileMenu.add(openTextItem);
52         fileMenu.add(saveTextItem);
53         fileMenu.add(exitItem);
54         menus.add(fileMenu);
55
56         openSerialItem.addActionListener(this);
57         saveSerialItem.addActionListener(this);
58         openTextItem.addActionListener(this);
59         saveTextItem.addActionListener(this);
60         exitItem.addActionListener(this);
61
62
```

```
63          actionMenu = new JMenu ("Action");
64          rentDVDItem = new JMenuItem("Rent DVD");
65          rentGameItem = new JMenuItem("Rent Game");
66          returnItem = new JMenuItem("Return");
67          searchTitleItem = new JMenuItem("Search Title");
68          dueDateItem = new JMenuItem("Search by Due Date");
69
70          actionMenu.add(rentDVDItem);
71          actionMenu.add(rentGameItem);
72          actionMenu.add(returnItem);
73          actionMenu.add(searchTitleItem);
74          actionMenu.add(dueDateItem);
75          menus.add(actionMenu);
76
77          rentDVDItem.addActionListener(this);
78          rentGameItem.addActionListener(this);
79          returnItem.addActionListener(this);
80          searchTitleItem.addActionListener(this);
81          dueDateItem.addActionListener(this);
82
83          myModel = new ListEngine();
84          myList = new JList(myModel);
85          myScroll = new JScrollPane(myList);
86          chooser = new JFileChooser();
87
88          this.setJMenuBar(menus);
89          this.add(myScroll);
90
91
92          this.setSize(700, 700);
93          this.setVisible(true);
94
95      }
96
97      public void actionPerformed(ActionEvent event) {
98          if(event.getSource() == exitItem)
99              System.exit(1);
100
101         if(event.getSource() == openSerialItem){
102             chooser.showOpenDialog(null);
103             try {
104                 FileInputStream fileIn =
105                 new FileInputStream(chooser.getSelectedFile());
106                 myModel.openSerialItem(fileIn);
107             } catch (FileNotFoundException e) {
108                 // TODO Auto-generated catch block
109                 e.printStackTrace();
110             } catch (IOException e) {
111                 // TODO Auto-generated catch block
112                 e.printStackTrace();
113             } catch (ClassNotFoundException e) {
114                 // TODO Auto-generated catch block
115                 e.printStackTrace();
116             } catch (NullPointerException e) {
117                 // TODO Auto-generated catch block
```

```
118
119             }
120
121         }
122
123
124         if(event.getSource() == saveSerialItem){
125             chooser.showSaveDialog(null);
126             try {
127                 FileOutputStream fileOut =
128                 new FileOutputStream(chooser.getSelectedFile());
129                 ObjectOutputStream out = new ObjectOutputStream(fileOut);
130                 myModel.saveSerialItem(out, fileOut);
131             } catch (FileNotFoundException e) {
132                 // TODO Auto-generated catch block
133                 e.printStackTrace();
134             } catch (IOException e) {
135                 // TODO Auto-generated catch block
136                 e.printStackTrace();
137             } catch (NullPointerException e) {
138                 // TODO Auto-generated catch block
139
140             }
141         }
142
143
144         if(event.getSource() == openTextItem){
145             chooser.showOpenDialog(null);
146             File file = chooser.getSelectedFile();
147             try {
148                 BufferedReader getInfo = new BufferedReader(new FileReader(file));
149                 myModel.openText(getInfo);
150
151             } catch (FileNotFoundException e) {
152                 // TODO Auto-generated catch block
153                 e.printStackTrace();
154             } catch (IOException e) {
155                 // TODO Auto-generated catch block
156                 e.printStackTrace();
157             } catch (ParseException e) {
158                 // TODO Auto-generated catch block
159                 e.printStackTrace();
160             } catch (NullPointerException e) {
161                 // TODO Auto-generated catch block
162
163             }
164         }
165
166
167         if(event.getSource() == saveTextItem){
168             chooser.showSaveDialog(null);
169         File file = chooser.getSelectedFile();
170         try {
171             PrintWriter filewriter = new PrintWriter(new BufferedWriter(new
    FileWriter(file)));
```

```
172             myModel.saveText(filewriter);
173         } catch (IOException e) {
174             // TODO Auto-generated catch block
175             e.printStackTrace();
176         } catch (NullPointerException e) {
177             // TODO Auto-generated catch block
178
179         }
180
181     }
182
183
184         if(event.getSource() == rentDVDItem) {
185             DVD d = new DVD();
186             DialogRentDVD dialog = new DialogRentDVD(this, d);
187             if(dialog.getClosedOk())
188             {
189                 myModel.addElement(dialog.getUnit());
190             }
191
192         }
193
194         if(event.getSource() == rentGameItem) {
195             Game game = new Game();
196             DialogRentGame dialog = new DialogRentGame(this, game);
197                 if(dialog.getClosedOk())
198                     myModel.addElement(dialog.getUnit());
199
200         }
201
202
203         if(event.getSource() == returnItem){
204             int selectedIndex = myList.getSelectedIndex();
205             String source = JOptionPane.showInputDialog(null, "Please enter a date.");
206             if(!source.isEmpty()){
207             if(selectedIndex != -1)
208                 myModel.deleteElement(selectedIndex, source);
209             }
210             else
211             {
212                 JOptionPane.showMessageDialog(null, "Sorry we need a date to complete
    search.");
213             }
214         }
215
216         if(event.getSource() == searchTitleItem){
217             CharSequence source = " ";
218             //JFrame input = new JFrame();
219             source = JOptionPane.showInputDialog("Search for for?");
220             myModel.search(source);
221         }
222
223         if(event.getSource() == dueDateItem){
224             String input =
225                 JOptionPane.showInputDialog("Search for what date?");
```

```
226             SimpleDateFormat simple = new
227                     SimpleDateFormat("MM/dd/yyyy");
228             Calendar x = new GregorianCalendar();
229             try {
230                 Date date = simple.parse(input);
231                 x.setTime(date);
232             } catch (ParseException  e) {
233                 // TODO Auto-generated catch block
234                 e.printStackTrace();
235             }
236             catch(NullPointerException e){
237
238             }
239
240             myModel.searchDate(x);
241
242
243         }
244
245     }
246
247 }
248
```

```java
package package1;

import java.io.*;

public class ListEngine extends AbstractListModel {

    /**ArracyList of DVD objects **/
    private ArrayList<DVD> listDVD;

    /*******************************************************
     Default constructor creates initializes ArrayList
     *******************************************************/
    public ListEngine(){

        listDVD = new ArrayList();
    }

    /**************************************
     Gets the object from the ArrayList
     @param the index of the item
     @return DVD object
     **************************************/
    @Override
    public Object getElementAt(int i) {

        return listDVD.get(i);
    }

    /**************************************
     Gets the object from the ArrayList
     @param the index of the item
     @return DVD object
     **************************************/
    @Override
    public int getSize() {

        return listDVD.size();
    }

    /**********************************
     Adds DVD object to the array.
     @param the index of the item
     **********************************/
    public void addElement(DVD d){
        listDVD.add(d);
        fireIntervalAdded(this, 0, listDVD.size());

    }

    /*******************************************************
     Removes the object from the array and update the
     JList. And calculates the rate for the rental.
     @param the index of the item and the date of the return.
     *******************************************************/
    public void deleteElement(int d, String input){
```

```
62
63          Calendar x = new GregorianCalendar();
64          SimpleDateFormat simple = new SimpleDateFormat("MM/dd/yyyy");
65          double priceOri = listDVD.get(d).getPirce();;
66          double total = 0.0;
67          double original = 0.0;
68          double priceLate = listDVD.get(d).getLatePirce();
69          NumberFormat formatter = NumberFormat.getCurrencyInstance();
70
71
72
73          try {
74              Date date = simple.parse(input);
75              x.setTime(date);
76          } catch (ParseException e) {
77           JOptionPane.showMessageDialog(null, "Sorry we'll we need at "+
78                                         "date please try again.");
79          }
80
81          //Check to make sure the entered date is after the rent date
82      if(listDVD.get(d).getRent().before(x)){
83
84          //Set the DVD values to variables
85          String name = listDVD.get(d).getNameOfRenter();
86          String title = listDVD.get(d).getTitle();
87          GregorianCalendar due = listDVD.get(d).getDue();
88          GregorianCalendar rent = listDVD.get(d).getRent();
89
90          //Checks if the return date is late
91          if(x.before(due) || x.equals(due)){
92          total = (diffDays(x.getTime(),
93          listDVD.get(d).getRent().getTime()) + 1) * priceOri;
94
95          }
96
97          else
98          {
99           //Caluclates the price before late charge
100          original = (diffDays(due.getTime(),rent.getTime()) + 1)
101                     * priceOri;
102          //Calculates and adds late charge
103          total = original + diffDays(x.getTime(), due.getTime()) *
104                  priceLate;
105         }
106
107         JOptionPane.showMessageDialog(null,"Thank you " + name +
108                         " for renting " + title + " you owe " +
109                         formatter.format(total));
110         listDVD.remove(d);
111         fireContentsChanged(this, 0, listDVD.size());
112         }
113
114      else
115      {
116
```

```
117             JOptionPane.showMessageDialog(null,"Please try a " +
118                                           "different date.");
119         }
120
121
122     }
123
124     /*****************************************************
125      Removes all objects from the array and update the
126      JList.
127      *****************************************************/
128     public void clearElements(){
129         this.listDVD.clear();
130         fireIntervalRemoved(this, 0, listDVD.size());
131
132     }
133
134
135     /*****************************************************
136      Save objects to a Serial file
137      *****************************************************/
138     public void saveSerialItem(ObjectOutputStream out,
139             FileOutputStream fileOut) throws IOException{
140
141         for(int i = 0; i < listDVD.size(); i++){
142             out.writeObject(listDVD.get(i));
143         }
144
145         out.close();
146         fileOut.close();
147
148     }
149
150     /*****************************************************
151      Loads the array from a serial file.
152      @param fileIn
153      *****************************************************/
154     public void openSerialItem(FileInputStream fileIn) throws
155     IOException, ClassNotFoundException{
156
157         InputStream buffer = new BufferedInputStream(fileIn);
158         ObjectInputStream in = new ObjectInputStream(buffer);
159
160         //Clears Jlist to show new database
161         this.clearElements();
162
163         while(in != null){
164             try{
165                 DVD dvd = null;
166
167                 dvd = (DVD) in.readObject();
168                 this.addElement(dvd);
169
170             }
171             catch(EOFException x){
```

```
172                 break;
173             }
174         }
175 
176         in.close();
177         fileIn.close();
178     }
179 
180     /**************************************************
181      Save objects to a text file
182      @param filewriter, text file
183      **************************************************/
184     public void saveText(PrintWriter filewriter){
185     //Prints the object info: "Name, Date..."
186         String line = " ";
187         String lineTwo = " ";
188         for(int i = 0; i < listDVD.size(); i++){
189             DVD dvdUnit = listDVD.get(i);
190             line = dvdUnit.getClass().getName() + "," +
191             dvdUnit.getNameOfRenter() + "," + dvdUnit.getTitle() + "," +
192             dvdUnit.getRentString() + "," + dvdUnit.getBackString();
193 
194             if(dvdUnit instanceof Game){
195                 lineTwo = line + "," + ((Game)dvdUnit).getPlayer();
196                 filewriter.println(lineTwo);
197             }
198 
199             else{
200 
201                 filewriter.println(line);
202             }
203 
204         }
205 
206         filewriter.close();
207 
208 
209     }
210 
211     /**************************************************
212      Loads objects to the array List from a text file
213      @param filereader, text file
214      **************************************************/
215     public void openText(BufferedReader filereader)
216             throws IOException, ParseException{
217 
218         this.clearElements();
219 
220         String info = filereader.readLine();
221 
222         while(info != null){
223             //Splits the elements in text to strings
224             String[] listInfo = info.split(",");
225 
226             //Checks if the object is a DVD or Game
```

```
227             if(listInfo[0].equals("package1.DVD"))
228             {
229                 DVD newDVD = new DVD();
230                 newDVD.setnameOfRenter(listInfo[1]);
231                 newDVD.setTitle(listInfo[2]);
232                 newDVD.setRented(listInfo[3]);
233                 newDVD.setDueDate(listInfo[4]);
234                 this.addElement(newDVD);
235             }
236
237
238             else
239             {
240                 Game newGame = new Game();
241                 newGame.setnameOfRenter(listInfo[1]);
242                 newGame.setTitle(listInfo[2]);
243                 newGame.setRented(listInfo[3]);
244                 newGame.setDueDate(listInfo[4]);
245                 newGame.setPlayer(PlayerType.valueOf(listInfo[5]));
246                 this.addElement(newGame);
247             }
248
249             info = filereader.readLine();
250         }
251
252     }
253     /*******************************************************
254      Searchs through the Array list to find all objects
255      contain the CharSequence
256      @param the sequence your searching for
257      *******************************************************/
258     public void search(CharSequence source){
259         try{
260
261             ArrayList<DVD> match = new ArrayList();
262
263             for(int i = 0; i < listDVD.size(); i++){
264                 //Fills an array list with matches
265                 if(listDVD.get(i).getTitle().contains(source))
266                     match.add(listDVD.get(i));
267             }
268
269             //Clears the elements
270             this.clearElements();
271
272             //Adds the elements found
273             for(int i = 0; i < match.size(); i++){
274
275                 this.addElement(match.get(i));
276             }
277         }
278
279         catch(NullPointerException e){
280
281         }
```

```
282
283
284     }
285
286     /*******************************************************
287     Finds rentals past due.
288     @param Due date
289     *******************************************************/
290    public void searchDate(Calendar due){
291     String output = "";
292     for(int i = 0; i < listDVD.size(); i++){
293     if(listDVD.get(i).getDue().getTime().after(due.getTime())){
294     int days =
295     diffDays(listDVD.get(i).getDue().getTime(), due.getTime());
296      output += listDVD.get(i).toString() +" "  + days + " days \n";
297        }
298             }
299             JOptionPane.showMessageDialog(null, output);
300
301     }
302     /*******************************************************
303      Calculates the number day between two dates.
304      I found this method on stackoverflow.com
305      @param Date d1, Date d2
306      @return the difference between d1 and d2
307      *******************************************************/
308     private int diffDays(Date d1, Date d2){
309
310         return (int)((d1.getTime() - d2.getTime()) /
311                 (1000 * 60 * 60 * 24));
312
313     }
314
315
316 }
317
```