

Marvellous File Packer Unpacker

```
1.// MarvellousMain.java
2.
3.import javax.swing.*;
4.import java.awt.*;
5.import java.awt.event.*;
6.import java.awt.BorderLayout;
7.import java.awt.Dimension;
8.import javax.swing.JLabel;
9.
10.class MarvellousLogin extends Template implements ActionListener, Runnable
11.{
12.    JButton SUBMIT;
13.    JLabel label1,label2,label3,TopLabel;
14.    final JTextField text1,text2;
15.    private static int attemp = 3;
16.
17.    MarvellousLogin()
18.    {
19.        TopLabel = new JLabel();
20.        TopLabel.setHorizontalAlignment(SwingConstants.CENTER);
21.        TopLabel.setText("Marvellous Packer Unpacker : Login");
22.        TopLabel.setForeground(Color.BLUE);
23.
24.        Dimension topsize = TopLabel.getPreferredSize();
25.        TopLabel.setBounds(50,40, topsize.width, topsize.height);
26.        _header.add(TopLabel);
27.
28.        label1 = new JLabel();
29.        label1.setText("Username:");
30.        label1.setForeground(Color.white);
31.
32.        Dimension size = label1.getPreferredSize();
33.
34.        label1.setBounds(50,135,size.width,size.height);
35.        label1.setHorizontalAlignment(SwingConstants.CENTER);
36.
37.        text1 = new JTextField(15);
38.        Dimension tsize = text1.getPreferredSize();
39.        text1.setBounds(200,135, tsize.width, tsize.height);
40.
41.        text1.setToolTipText("ENTER USERNAME");
42.
43.        label2 = new JLabel();
44.        label2.setText("Password:");
45.        label2.setBounds(50,175, size.width, size.height);
46.        label2.setForeground(Color.white);
47.        label2.setHorizontalAlignment(SwingConstants.CENTER);
48.
49.        text2 = new JPasswordField(15);
50.        text2.setBounds(200,175, tsize.width, tsize.height);
```

```

48. text2 = new JPasswordField(15);
49. text2.setBounds(200,175, tsize.width, tsize.height);
50.
51. text2.setToolTipText("ENTER PASSWORD");
52.
53. text2.addFocusListener(new FocusListener()
54. {
55.     public void focusGained(FocusEvent e)
56.     {
57.
58.     }
59.     public void focusLost(FocusEvent e)
60.     {
61.         label3.setText("");
62.     }
63. });
64.
65. SUBMIT=new JButton("SUBMIT");
66. SUBMIT.setHorizontalAlignment(SwingConstants.CENTER);
67.
68. Dimension ssize = SUBMIT.getPreferredSize();
69.
70. SUBMIT.setBounds(50,220,ssize.width,ssize.height );
71.
72. label3 = new JLabel();
73. label3.setText("");
74.
75. Dimension l3size = label3.getPreferredSize();
76.
77. label3.setForeground(Color.RED);
78. label3.setBounds(50,250 , l3size.width , l3size.height);
79.
80. Thread t = new Thread(this);
81. t.start();
82.
83. _content.add(label1);
84. _content.add(text1);
85.
86. _content.add(label2);
87. _content.add(text2);
88.
89. _content.add(label3);
90. _content.add(SUBMIT);
91.
92. pack();
93. validate();
94.
95. ClockHome();
96. setVisible(true);
97. this.setSize(500,500);
98. this.setResizable(false);
99. setLocationRelativeTo(null);
100. SUBMIT.addActionListener(this);
101.
102. }

```

```
103. public boolean Validate(String Username, String password)
104. {
105.     if((Username.length()<8)||((password.length() < 8))
106.         return false;
107.     else
108.         return true;
109. }
110.
111. public void actionPerformed(ActionEvent ae)
112. {
113.     String value1=text1.getText();
114.     String value2=text2.getText();
115.
116.     if ( ae.getSource() == exit )
117.     {
118.         this.setVisible(false);
119.         System.exit(0);
120.     }
121.
122.     if ( ae.getSource() == minimize )
123.     {
124.         this.setState(this.ICONIFIED);
125.     }
126.
127.     if(ae.getSource()==SUBMIT)
128.     {
129.         if(Validate(value1,value2) == false)
130.         {
131.             text1.setText("");
132.             text2.setText("");
133.             JOptionPane.showMessageDialog(this, "Short username", "Marvellous Packer Unpacker", JOptionPane.ERROR_MESSAGE);
134.         }
135.         if (value1.equals("MarvellousAdmin") && value2.equals("MarvellousAdmin"))
136.         {
137.             NextPage page = new NextPage(value1);
138.             this.setVisible(false);
139.             page.pack();
140.             page.setVisible(true);
141.             page.setSize(500, 500);
142.         }
143.         else
144.         {
145.             attemp--;
146.
147.             if(attemp == 0)
148.             {
149.                 JOptionPane.showMessageDialog(this, "Number of attempts finished", "Marvellous Packer Unpacker", JOptionPane.ERROR_MESSAGE);
150.                 this.dispose();
151.                 System.exit(0);
152.             }
153.
154.             JOptionPane.showMessageDialog(this, "Incorrect login or password",
155.                 "Error", JOptionPane.ERROR_MESSAGE);
156.         }
157.     }
158. }
```

```

158. }
159.
160. public void run()
161. {
162.     for(;;)
163.     {
164.         if(text2.isFocusOwner())
165.         {
166.             if( Toolkit.getDefaultToolkit().getLockingKeyState (KeyEvent.VK_CAPS_LOCK ) )
167.             {
168.                 text2.setToolTipText("Warning : CAPS LOCK is on");
169.             }
170.             else
171.                 text2.setToolTipText("");
172.
173.             if((text2.getText()).length() < 8)
174.                 label3.setText("Weak Password");
175.             else
176.                 label3.setText("");
177.         }
178.     }
179. }
180. }
181.
182. class MarvellousMain
183. {
184.     public static void main(String arg[])
185.     {
186.         try
187.         {
188.             MarvellousLogin frame=new MarvellousLogin();
189.             frame.setVisible(true);
190.         }
191.         catch(Exception e)
192.         {
193.             JOptionPane.showMessageDialog(null, e.getMessage());
194.         }
195.     }
196. }
197. // Template.java
198.
199. import javax.swing.*.*;
200. import javax.swing.plaf.basic.BasicBorders;
201. import java.awt.*.*;
202. import java.awt.event.ActionEvent;
203. import java.awt.event.ActionListener;
204. import java.io.Serializable;
205. import java.text.SimpleDateFormat;
206. import java.util.Date;

```

```
207.class ClockLabel extends JLabel implements ActionListener
208.{
209.    String type;
210.    SimpleDateFormat sdf;
211.
212.    public ClockLabel(String type)
213.    {
214.        this.type = type;
215.        setForeground(Color.green);
216.
217.        switch (type)
218.        {
219.            case "date" : sdf = new SimpleDateFormat(" MMMM dd yyyy");
220.                setFont(new Font("sans-serif", Font.PLAIN, 12));
221.                setHorizontalAlignment(SwingConstants.LEFT);
222.                break;
223.            case "time" : sdf = new SimpleDateFormat("hh:mm:ss a");
224.                setFont(new Font("sans-serif", Font.PLAIN, 40));
225.                setHorizontalAlignment(SwingConstants.CENTER);
226.                break;
227.            case "day" : sdf = new SimpleDateFormat("EEEE ");
228.                setFont(new Font("sans-serif", Font.PLAIN, 16));
229.                setHorizontalAlignment(SwingConstants.RIGHT);
230.                break;
231.            default : sdf = new SimpleDateFormat();
232.                break;
233.        }
234.
235.        Timer t = new Timer(1000, this);
236.        t.start();
237.    }
238.
239.    public void actionPerformed(ActionEvent ae)
240.    {
241.        Date d = new Date();
242.        setText(sdf.format(d));
243.    }
244.}
245.
246.class Template extends JFrame implements Serializable , ActionListener
247.{
248.    JPanel _header;
249.    JPanel _content;
250.    JPanel _top;
251.
252.    ClockLabel dayLabel;
253.    ClockLabel timeLabel;
254.    ClockLabel dateLabel;
255.}
```



```

251. ClockLabel dayLabel;
252. ClockLabel timeLabel;
253. ClockLabel dateLabel;
254.
255.
256. JButton minimize , exit;
257.
258. public Template()
259. {
260.     setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
261.     GridBagLayout grid = new GridBagLayout();
262.     setLayout(grid);
263.
264.     _top = new JPanel();
265.     _top.setBackground(Color.LIGHT_GRAY);
266.
267.     _top.setLayout(null);
268.
269.     getContentPane().add(_top,new GridBagConstraints(0,0,1,1,1,5,GridBagConstraints.BASELINE,GridBagConstraints.BOTH,new
Insets(0,0,0,0),0,0));
270.
271.     _header = new JPanel();
272.     _header.setLayout(null);
273.
274.     _header.setBackground(Color.white);
275.
276.     getContentPane().add(_header,new GridBagConstraints(0,1,1,1,1,20,GridBagConstraints.BASELINE,GridBagConstraints.BOTH,new
Insets(0,0,0,0),0,0));
277.
278.     _content = new JPanel();
279.     _content.setLayout(null);
280.     _content.setBackground(new Color(0,50,120));
281.     JScrollPane jsp = new JScrollPane(_content,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
282.     jsp.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
283.
284.     getContentPane().add(jsp,new GridBagConstraints(0,2,1,1,1,75,GridBagConstraints.BASELINE,GridBagConstraints.BOTH,new Insets(0,0,0,0),0,0));
285.     setTitle("Marvellous Packer-Unpacker");
286.
287.     Clock();
288.     CloseAndMin();
289. }
290.
291. void CloseAndMin()
292. {
293.     minimize=new JButton("-");
294.     minimize.setBackground(Color.LIGHT_GRAY);
295.     minimize.setBounds(MAXIMIZED_HORIZ,0,45,20 );
296.
297.     exit=new JButton("X");
298.     exit.setHorizontalAlignment(SwingConstants.CENTER);
299.     exit.setBackground(Color.LIGHT_GRAY);
300.     exit.setHorizontalTextPosition(0);
301.     exit.setBounds(MAXIMIZED_HORIZ+45,0,45,20 );
302.
303.     _top.add(minimize);
304.     _top.add(exit);
305.
306.     exit.addActionListener(this);
307.     minimize.addActionListener(this);
308. }

```

```
309. public void actionPerformed(ActionEvent ae)
310. {
311.     if ( ae.getSource() == exit )
312.     {
313.         this.setVisible(false);
314.         System.exit(0);
315.     }
316.
317.     if ( ae.getSource() == minimize )
318.     {
319.         setState(JFrame.ICONIFIED);
320.     }
321. }
322.
323. void Clock ()
324. {
325.     dateLabel = new ClockLabel("date");
326.     timeLabel = new ClockLabel("time");
327.     dayLabel = new ClockLabel("day");
328.
329.     dateLabel.setForeground (Color.blue);
330.     timeLabel.setForeground (Color.blue);
331.     dayLabel.setForeground (Color.blue);
332.
333.     dayLabel.setFont(new Font("Century",Font.BOLD,15));
334.
335.     dayLabel.setBounds(700,10,200, 100);
336.
337.     dateLabel.setFont(new Font("Century",Font.BOLD,15));
338.
339.     dateLabel.setBounds(800,-40,200, 100);
340.
341.     timeLabel.setFont(new Font("Century",Font.BOLD,15));
342.
343.     timeLabel.setBounds(760,-15,200, 100);
344.
345.     _header.add(dateLabel);
346.     _header.add(timeLabel);
347.     _header.add(dayLabel);
348. }
349.
350. void ClockHome()
351. {
352.     dateLabel = new ClockLabel("date");
353.     timeLabel = new ClockLabel("time");
354.     dayLabel = new ClockLabel("day");
355.
356.     dateLabel.setForeground (Color.blue);
357.     timeLabel.setForeground (Color.blue);
358.     dayLabel.setForeground (Color.blue);
359.     dayLabel.setFont(new Font("Century",Font.BOLD,15));
360.     dayLabel.setBounds(200,20,200, 100);
361.     dateLabel.setFont(new Font("Century",Font.BOLD,15));
362.     dateLabel.setBounds(300,-40,200, 100);
363.
364.     timeLabel.setFont(new Font("Century",Font.BOLD,15));
365.     timeLabel.setBounds(260,-10,200, 100);
366.
367.
```

```

367.
368.     _header.add(dateLabel);
369.     _header.add(timeLabel);
370.     _header.add(dayLabel);
371. }
372.}
373.
374.//NextPage.java
375.
376.import javax.swing.*;
377.import java.awt.*;
378.import java.awt.event.ActionEvent;
379.import java.awt.event.ActionListener;
380.
381.class NextPage extends Template implements ActionListener
382.{
383.    JLabel label;
384.    JButton pack , unpack;
385.
386.    NextPage(String value)
387.    {
388.        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
389.
390.        label = new JLabel("Welcome: "+value);
391.        Dimension size = label.getPreferredSize();
392.        label.setBounds(40,50, size.width + 60, size.height);
393.        label.setFont(new Font("Century",Font.BOLD,17));
394.        label.setForeground (Color.blue);
395.
396.        pack=new JButton("Pack Files");
397.        Dimension bsize = pack.getPreferredSize();
398.        pack.setBounds(100,100, bsize.width, bsize.height);
399.        pack.addActionListener(this);
400.
401.        unpack=new JButton("Unpack Files");
402.        Dimension b2size = unpack.getPreferredSize();
403.        unpack.setBounds(300,100, b2size.width, b2size.height);
404.        unpack.addActionListener(this);
405.
406.        _header.add(label);
407.        _content.add(pack);
408.        _content.add(unpack);
409.
410.        ClockHome();
411.        this.setSize(600,600);
412.        this.setResizable(false);

```



```

413.     this.setVisible(true);
414. }
415.
416. public void actionPerformed(ActionEvent ae)
417. {
418.     if ( ae.getSource() == exit )
419.     {
420.         this.setVisible(false);
421.         System.exit(0);
422.     }
423.     if ( ae.getSource() == minimize )
424.     {
425.         this.setState(this.ICONIFIED);
426.     }
427.     if ( ae.getSource() == pack )
428.     {
429.         this.setVisible(false);
430.         try
431.         {
432.             MarvellousPackFront obj = new MarvellousPackFront();
433.         }
434.         catch(Exception e){}
435.     }
436.     if ( ae.getSource() == unpack )
437.     {
438.         this.setVisible(false);
439.         MarvellousUnpackFront obj = new MarvellousUnpackFront();
440.     }
441. }
442.
443. //MarvellousPackFront.java
444.
445. import javax.swing.*;
446. import java.awt.*;
447. import java.awt.event.ActionEvent;
448. import java.awt.event.ActionListener;
449.
450. public class MarvellousPackFront extends Template implements ActionListener
451. {
452.     JButton SUBMIT ,PREVIOUS;
453.     JLabel label1,label2, title ;
454.     final JTextField text1,text2 ;
455.
456.     public MarvellousPackFront()
457.     {
458.         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
459.
460.         title = new JLabel("Marvellous Packing Portal");
461.         Dimension size = title.getPreferredSize();
462.         title.setBounds(40,50, size.width + 60, size.height);
463.         title.setFont(new Font("Century",Font.BOLD,17));
464.         title.setForeground (Color.blue);
465.
466.         label1 = new JLabel();
467.         label1.setText("Directory name");
468.         label1.setForeground(Color.white);
469.         label1.setBounds(350,50, size.width, size.height);
470.
471.         text1 = new JTextField(15);
472.         Dimension tsize = text1.getPreferredSize();

```

```

472. Dimension tsize = text1.getPreferredSize();
473. text1.setBounds(500,50, tsize.width, tsize.height);
474. text1.setToolTipText("Enter name of directory ");
475.
476. label2 = new JLabel();
477. label2.setText("Destination file name");
478. label2.setForeground(Color.white);
479. label2.setBounds(350,100, size.width + 60, size.height);
480.
481. text2 = new JTextField(15);
482. text2.setBounds(500,100, tsize.width, tsize.height);
483. text2.setToolTipText("Enter Destination file name");
484.
485. SUBMIT=new JButton("SUBMIT");
486. Dimension bsize = SUBMIT.getPreferredSize();
487. SUBMIT.setBounds(350,200, bsize.width, bsize.height);
488. SUBMIT.addActionListener(this);
489.
490. PREVIOUS = new JButton("PREVIOUS");
491. Dimension b2size = PREVIOUS.getPreferredSize();
492. PREVIOUS.setBounds(500, 200, b2size.width, b2size.height);
493. PREVIOUS.addActionListener(this);
494.
495. _header.add(title);
496. _content.add(label1);
497. _content.add(label2);
498. _content.add(text1);
499. _content.add(text2);
500. _content.add(SUBMIT);
501. _content.add(PREVIOUS);
502.
503. this.setSize(1000,400);
504. this.setResizable(false);
505. this.setVisible(true);
506. text1.requestFocusInWindow();
507. }
508.
509. public void actionPerformed(ActionEvent ae)
510. {
511.     if ( ae.getSource() == exit )
512.     {
513.         this.setVisible(false);
514.         System.exit(0);
515.     }
516.

```

```

517.     if ( ae.getSource() == minimize )
518.     {
519.         this.setState(this.ICONIFIED);
520.     }
521.     if ( ae.getSource() == SUBMIT )
522.     {
523.         try
524.         {
525.             MarvellousPacker obj = new MarvellousPacker(text1.getText(),text2.getText());
526.             this.dispose();
527.             NextPage t = new NextPage("MarvellousAdmin");
528.         }
529.         catch(Exception e){}
530.     }
531.     if ( ae.getSource() == PREVIOUS )
532.     {
533.         this.setVisible(false);
534.         this.dispose();
535.         NextPage t = new NextPage("MarvellousAdmin");
536.     }
537. }
538.}
539.
540.// MarvellousPacker.java
541.
542.import java.io.BufferedReader;
543.import java.io.File;
544.import java.io.FileReader;
545.import java.io.IOException;
546.import java.nio.file.Files;
547.import java.nio.file.Path;
548.import java.nio.file.Paths;
549.import java.util.List;
550.import java.util.stream.Stream;
551.import java.io.File;
552.import java.io.FileInputStream;
553.import java.io.FileOutputStream;
554.import java.io.IOException;
555.import java.util.Arrays;
556.
557.public class MarvellousPacker
558.{
559.    FileOutputStream outstream = null;
560.
561.    String ValidExt[] = {".txt",".c",".java",".cpp"};
562.
563.    public MarvellousPacker(String src, String Dest) throws Exception
564.    {
565.        String Magic = "Marvellous11";
566.        byte arr[] = Magic.getBytes();
567.        File outfile = new File(Dest);
568.
569.        File infile = null;
570.        outstream = new FileOutputStream(Dest);
571.        outstream.write(arr, 0, arr.length);
572.
573.        File folder = new File(src);
574.
575.        System.setProperty("user.dir",src);
576.

```

```
575. System.setProperty("user.dir",src);
576.
577. listAllFiles(src);
578. }
579.
580. public void listAllFiles(String path)
581. {
582.     try
583.     (Stream<Path> paths = Files.walk(Paths.get(path)))
584.     {
585.         paths.forEach(filePath ->
586.             {
587.                 if (Files.isRegularFile(filePath))
588.                 {
589.                     try
590.                     {
591.                         String name = filePath.getFileName().toString();
592.                         String ext = name.substring(name.lastIndexOf("."));
593.
594.                         List<String> list = Arrays.asList(ValidExt);
595.
596.                         if(list.contains(ext))
597.                         {
598.                             File file=new File(filePath.getFileName().toString());
599.
600.                             Pack(file.getAbsolutePath());
601.                         }
602.                     }
603.                     catch (Exception e)
604.                     {
605.                         System.out.println(e);
606.                     }
607.                 }
608.             });
609.     }
610.     catch (IOException e)
611.     {
612.         System.out.println(e);
613.     }
614. }
615.
616. public void Pack(String filePath)
617. {
618.     FileInputStream instream = null;
619.
620.
```

```

621. try
622. {
623.     byte[] buffer = new byte[1024];
624.
625.     int length;
626.
627.     byte temp[] = new byte[100];
628.
629.     File fobj = new File(filePath);
630.
631.     String Header = filePath+" "+fobj.length();
632.
633.     for (int i = Header.length(); i < 100; i++)
634.         Header += " ";
635.
636.     temp = Header.getBytes();
637.
638.     instream = new FileInputStream(filePath);
639.
640.     outstream.write(temp, 0, temp.length);
641.
642.     while ((length = instream.read(buffer)) > 0)
643.     {
644.         outstream.write(buffer, 0, length);
645.     }
646.
647.     instream.close();
648. }
649. catch(Exception e)
650. {
651.     System.out.println(e);
652. }
653. }
654. }
655.
656. // MarvellousUnpackFront.java
657.
658. import javax.swing.*;
659. import java.awt.*;
660. import java.awt.event.ActionEvent;
661. import java.awt.event.ActionListener;
662.
663. class InvalidFileException extends Exception
664. {
665.     public InvalidFileException(String str)
666.     {
667.         super(str);
668.     }
669. }
670.
671. public class MarvellousUnpackFront extends Template implements ActionListener
672. {
673.     JButton SUBMIT ,PREVIOUS;
674.     JLabel label1,label2, title ;
675.     final JTextField text1 ;
676.
677.     public MarvellousUnpackFront()
678.     {
679.
680.         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

```



```

679.
680.     setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
681.
682.     title = new JLabel("UnPacking Portal");
683.     Dimension size = title.getPreferredSize();
684.     title.setBounds(40,50, size.width + 60, size.height);
685.     title.setFont(new Font("Century",Font.BOLD,17));
686.     title.setForeground (Color.blue);
687.
688.     label1 = new JLabel();
689.     label1.setText("File Name");
690.     label1.setForeground(Color.white);
691.     label1.setBounds(350,50, size.width, size.height);
692.
693.     text1 = new JTextField(15);
694.     Dimension tsize = text1.getPreferredSize();
695.     text1.setBounds(500,50, tsize.width, tsize.height);
696.     text1.setToolTipText("Enter name of directory ");
697.
698.     SUBMIT=new JButton("Extract Here");
699.     Dimension bsize = SUBMIT.getPreferredSize();
700.     SUBMIT.setBounds(350,200, bsize.width, bsize.height);
701.     SUBMIT.addActionListener(this);
702.
703.     PREVIOUS = new JButton("PREVIOUS");
704.     Dimension b2size = PREVIOUS.getPreferredSize();
705.     PREVIOUS.setBounds(500, 200, b2size.width, b2size.height);
706.     PREVIOUS.addActionListener(this);
707.
708.     _header.add(title);
709.     _content.add(label1);
710.     _content.add(text1);
711.     _content.add(SUBMIT);
712.     _content.add(PREVIOUS);
713.
714.     this.setSize(1000,400);
715.     this.setResizable(false);
716.     this.setVisible(true);
717.     text1.requestFocusInWindow();
718. }
719.
720. public void actionPerformed(ActionEvent ae)
721. {
722.     if ( ae.getSource() == exit )
723.     {
724.

```

```

725.         this.setVisible(false);
726.         System.exit(0);
727.     }
728.     if ( ae.getSource() == minimize )
729.     {
730.         this.setState(this.ICONIFIED);
731.     }
732.     if ( ae.getSource() == SUBMIT )
733.     {
734.         try{
735.             MarvellousUnpack obj = new MarvellousUnpack(text1.getText());
736.             this.dispose();
737.             NextPage t = new NextPage("admin");
738.         }
739.         catch(InvalidFileException obj)
740.         {
741.             this.setVisible(false);
742.             this.dispose();
743.
744.             JOptionPane.showMessageDialog(this, "Invalid Packed File",
745.                 "Error", JOptionPane.ERROR_MESSAGE);
746.
747.             NextPage t = new NextPage("MarvellousAdmin");
748.         }
749.         catch(Exception e)
750.         {
751.         }
752.     }
753.     if ( ae.getSource() == PREVIOUS )
754.     {
755.         this.setVisible(false);
756.         this.dispose();
757.         NextPage t = new NextPage("admin");
758.     }
759. }
760.
761. //MarvellousUnpack.java
762.
763. import java.io.BufferedReader;
764. import java.io.File;
765. import java.io.FileReader;
766. import java.io.IOException;
767. import java.nio.file.Files;
768. import java.nio.file.Path;
769. import java.nio.file.Paths;
770. import java.io.FileInputStream;
771. import java.io.FileOutputStream;
772.
773. public class MarvellousUnpack
774. {
775.     FileOutputStream outstream = null;
776.
777.     public MarvellousUnpack(String src) throws Exception
778.     {
779.         unpack(src);
780.     }
781.
782.     public void unpack(String filePath) throws Exception

```

```

780. }
781.
782. public void unpack(String filePath) throws Exception
783. {
784.     try
785.     {
786.         FileInputStream instream = new FileInputStream(filePath);
787.
788.         byte header[] = new byte[100];
789.         int length = 0;
790.
791.         byte Magic[] = new byte[12];
792.         instream.read(Magic,0,Magic.length);
793.
794.         String Magicstr = new String(Magic);
795.
796.         if(!Magicstr.equals("Marvellous11"))
797.         {
798.             throw new InvalidFileException("Invalid packed file format");
799.         }
800.
801.         while((length = instream.read(header,0,100)) > 0)
802.         {
803.             String str = new String(header);
804.
805.             String ext = str.substring(str.lastIndexOf("/"));
806.             ext = ext.substring(1);
807.
808.             String[] words = ext.split("\\s");
809.
810.             String filename = words[0];
811.
812.             int size = Integer.parseInt(words[1]);
813.
814.             byte arr[] = new byte[size];
815.
816.             instream.read(arr,0,size);
817.
818.             FileOutputStream fout = new FileOutputStream(filename);
819.             fout.write(arr,0,size);
820.         }
821.     }
822.     catch(InvalidFileException obj)
823.     {
824.         throw new InvalidFileException("Invalid packed file format");
825.     }
826.     catch(Exception e)
827.     {}
828. }

```