Chapter 10 - Events Handling

Question 1:

True or False:

It is possible to use both the delegation model and the Java 1.0 event model together in the same program.

[] A) true

[] B) false

Question 2:

Which statement or statements are true about the code fragment listed below ?

```
1.  class MyButton extends Button
2.  {
3.      public MyButton(String str)
4.      {
5.          super(str);
6.          enableEvents(AWT.ACTION_EVENT_MASK);
7.      }
8.      public void processActionEvent(ActionEvent actionEvent)
```

```
9.              {
10.                 System.out.println("HOLAHOLAHOLAHUPE");
11.             }
12.         }
```

[] A) Creating a new instance of the MyButton class using the constructor at line 3 will produce a new button that its action events won"t be sent to the listener.

[] B) By deleting line number 6, creating a new instance of the MyButton class using the constructor at line 3 will produce a new button that its action events will be sent to the listener.

[] C) Calling the enableEvents method might limit the number of listeners.

Question 3:

True or False:

It is possible to use the 1.0 event model in a program that runs under the 2.0 JVM.

[] A) True

[] B) False

Question 4:

True or False:

It is possible to create a class that its objects will be able to be listeners to several event types.

[] A) True

[] B) False

Question 5:

True or False:

As in the 1.0 event model, in the delegation model the methods that are activated as an events happen also return a Boolean value.

[] A) True

[] B) False

Question 6:

True or False:

A component can have an unlimited number of listeners to the same type of event.

[] A) True

[] B) False

Question 7:

True or False:

When an object has several event listeners there is no order in their invocation.

[] A) True

[] B) False

Question 8:

Given the code below:

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class Framy extends Frame
4.  {
5.      public static void main(String args[])
6.      {
7.          Framy be = new Framy();
8.          be.setSize(200,200);
9.          be.setVisible(true);
10.     }
11.     public Framy()
12.     {
13.         super("Framy");
14.         b = new Button("PRESS");
15.         add(b);
16.         b.addActionListener(this);
17.     }
18.     public void actionPerformed(ActionEvent ae)
19.     {
20.         System.exit(0);
21.     }
```

```
22.         private Button b;
23. }
```

which of the following sentences (one or more) is true ?

[] A) The program displays a frame with a big button inside of it.

[] B) Pressing the button that will be displayed will cause the termination of the program.

[] C) The compilation fails in line 15.

[] D) The compilation fails in line 16.

[] E) The compilation succeed but the execution fails.

Question 9:

Given the code below:

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  import java.applet.*;
4.  public class Mac extends Applet implements ActionListener
5.  {
6.          TextField tf;
7.          Button b1,b2;
```

```
8.          public void init()
9.          {
10.                 setLayout(new GridLayout());
11.                 add(b1=new Button("One"));
12.                 b1.addActionListener(this);
13.                 add(b2=new Button("Two"));
14.                 b2.addActionListener(this);
15.                 add(tf = new TextField());
16.         }
17.         public void actionPerformed(ActionEvent e)
18.         {
19.                 tf.setText(e.getActionCommand());
20.         }
21.         public static void main(String args[])
22.         {
23.                 Frame f = new java.awt.Frame("zulu");
24.                 Panel peny = new Mac();
25.                 ((Applet)peny).init();
26.                 f.add(peny);
27.                 f.setSize(400,300);
28.                 f.setVisible(true);
29.         }
30. }
```

[] A) The code doesn"t compile

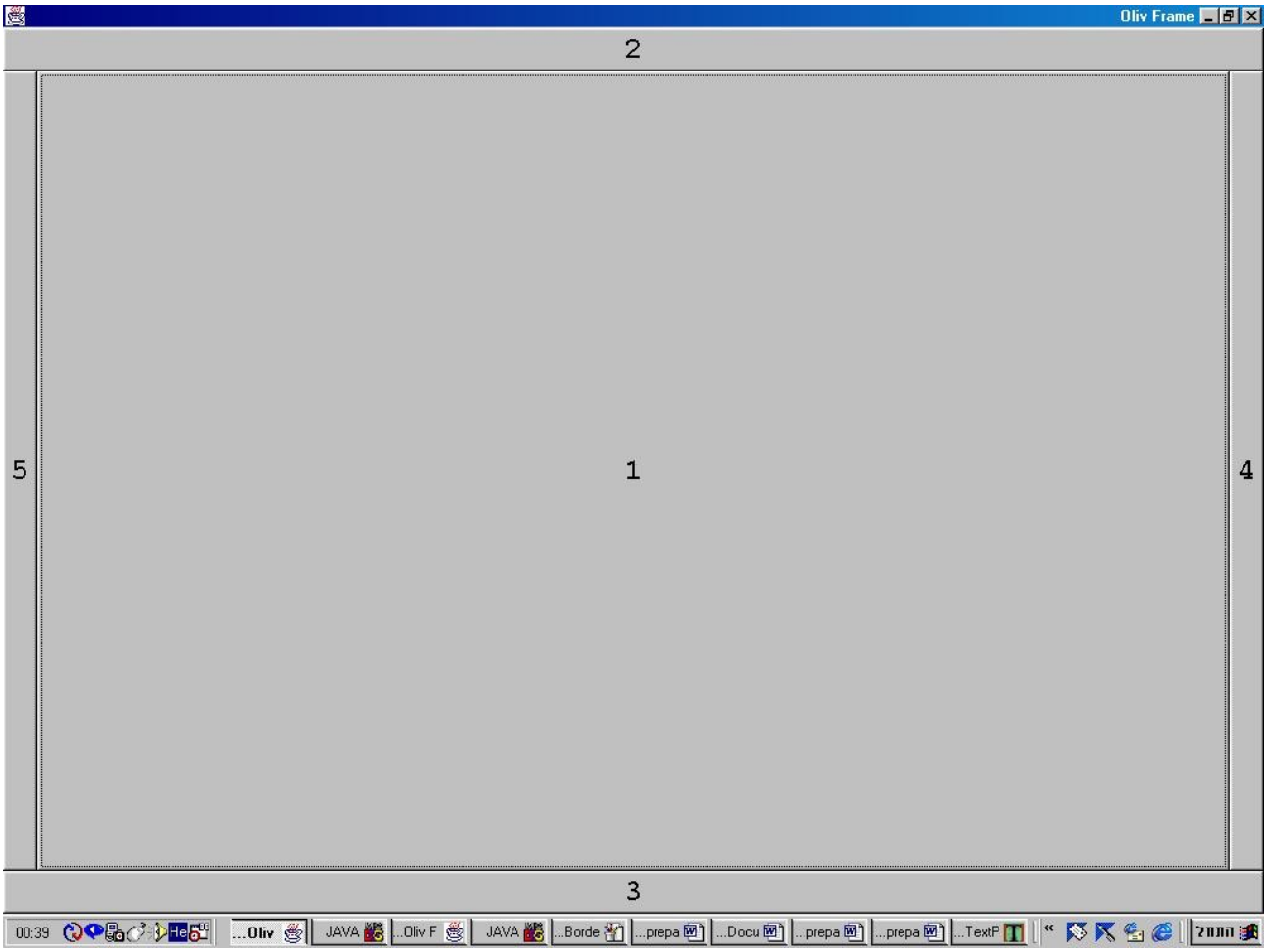
[] B) The code compiles successfully

[] C) The code creates a Frame with two buttons

[] D) As the code executes successfully, pushing each one of the buttons displays the text, which is written on the pushed button, in the text field.

Question 10:

Given the GUI screen below:



The code below:

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class OlivFrame extends Frame
4.  {
5.          public static void main(String args[])
6.          {
7.                  OlivFrame frame = new OlivFrame("Oliv Frame");
8.                  frame.go();
9.          }

10.         public OlivFrame(String str)
11.         {
12.                 super(str);
13.         }

14.         public void go()
15.         {
16.                 setLayout(new BorderLayout());
17.                 Button buttons[] = new Button[5];
18.                 String position[] = {"Center", "North", "South", "East", "West"};
19.                 for(int i=0; i<buttons.length; i++)
20.                 {
21.                         buttons[i] = new Button(""+(i+1));
```

```
22.                 buttons[i].setFont(new Font("Courier", Font.BOLD, 24));
23.                 add(buttons[i], position[i]);
24.             }
25.             addWindowListener(_____________________
26.             {
27.                 public void windowClosing(WindowEvent event)
28.                 {
29.                     System.exit(0);
30.                 }
31.             }                                                    );
32.             setSize(500,500);
33.             setVisible(true);
34.         }
35. }
```

The missing line might be (one or more):

[] A) new WindowAdapter()

[] B) new WindowListener()

Question 11:

Given the following code:

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.
4.
5.
6.
7.  public class ConsumeDemo
8.  {
9.          Frame frame;
10.         TextField tf;
11.
12.         public ConsumeDemo()
13.         {}
14.
15.         public void go()
16.         {
17.                 frame = new Frame("ConsumeDemo");
18.                 tf = new NumbersTextField(10);
19.                 frame.setLayout(new FlowLayout());
20.                 frame.add(tf);
21.                 frame.addWindowListener(new WindowAdapter()
```

```
22.                     {
23.                             public void windowClosing(WindowEvent event)
24.                             {
25.                                     System.exit(0);
26.                             }
27.                     }                       );
28.             frame.pack();
29.             frame.setVisible(true);
30.
31.     }
32.
33.     public static void main(String args[])
34.     {
35.             ConsumeDemo demo = new ConsumeDemo();
36.             demo.go();
37.     }
38. }
39.
40.
41. class NumbersTextField extends TextField implements KeyListener
42. {
43.
44.     NumbersTextField(int num)
45.     {
46.             super(num);
```

```
47.             this.addKeyListener(this);
48.         }
49.
50.         NumbersTextField()
51.         {
52.             this(10);
53.         }
54.
55.         public void keyTyped(KeyEvent event)
56.         {
57.             if(event.isActionKey())
58.             {
59.                 return;
60.             }
61.             if(Character.isDigit(event.getKeyChar()))
62.             {
63.                 return;
64.             }
65.             event.consume();
66.         }
67.
68.         public void keyPressed (KeyEvent event) {}
69.         public void keyReleased (KeyEvent event) {}
70. }
```

() A) It will be possible typing every key

() B) It will be possible typing only digits

() C) It won"t be possible typing digits

Question 12:

True or False:

Input events are delivered to listeners before they are processed normally by the source where they originated. This allows listeners and component subclasses to "consume" the events so that the source will not process them in their default manner. In order to "consume" the event, a key listener should be set, and the consume method should be invoked within the keyTyped method.

() A) True

() B) False

Question 13:

True or False:

The text field that the following code presents doesn"t enable the user typing in to it.

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.
4.
5.  public class ConsumeDemo implements KeyListener
6.  {
7.      Frame frame;
8.      TextField tf;
9.
10.     public ConsumeDemo()
11.     {}
12.
13.     public void go()
14.     {
15.         frame = new Frame();
16.         tf = new TextField(10);
17.         tf.addKeyListener(this);
18.         frame.setLayout(new FlowLayout());
19.         frame.add(tf);
```

```
20.                     frame.addWindowListener(new WindowAdapter()
21.                     {
22.                             public void windowClosing(WindowEvent event)
23.                             {
24.                                     System.exit(0);
25.                             }
26.                     }                               );
27.                     frame.pack();
28.             frame.setVisible(true);
29.
30.     }
31.
32.     public void keyTyped(KeyEvent event)
33.     {
34.             event.consume();
35.     }
36.
37.     public void keyPressed (KeyEvent event) {}
38.     public void keyReleased (KeyEvent event) {}
39.
40.     public static void main(String args[])
41.     {
42.             ConsumeDemo demo = new ConsumeDemo();
43.             demo.go();
44.             }
```

45. }

() A) True

() B) False