5 Swing & AWT interview questions and answers

Posted on March 6, 2015 by Arulkumaran Kumaraswamipillai

If you are going to work on Swing based applications….

Q1. What are the differences between AWT and Swing?
A1. Swing provides a richer set of components than AWT. They are 100% Java-based. There are a few other advantages to Swing over AWT:

- Swing provides both additional components like JTable, JTree etc and added functionality to AWT-replacement components.
- Swing components can change their appearance based on the current “look and feel” library that’s being used.
- Swing components follow the Model-View-Controller (MVC) paradigm, and thus can provide a much more

flexible UI.

- Swing provides “extras” for components, such as: icons on many components, decorative borders for components, tool tips for components etc.
- Swing components are lightweight (less resource intensive than AWT).
- Swing provides built-in double buffering (which means an off-screen buffer [image] is used during drawing and then the resulting bits are copied onto the screen. The resulting image is smoother, less flicker and quicker than drawing directly on the screen).
- Swing provides paint debugging support for when you build your own component i.e.slow motion rendering.

Q2. How will you go about building a Swing GUI client?
A2. The steps involved in building a Swing GUI are:

Step 1: Firstly, you need a container like a Frame, a Window, or an Applet to display components like panels, buttons, text areas etc. The job of a container is to hold and display components. A container is also a component (note: uses a composite design pattern). A JPanel is a container as well.

```
1  import javax.swing.JFrame;
2  import javax.swing.JTextArea;
3
4  public class MyFrame extends JFrame {
5      public static void main(String[] args) {
6          JFrame frame = new JFrame("Frame Title")
7          ...// rest of the code to follow
8      }
9  }
10
```

Step 2: Create some components such as panels, buttons, text areas etc.

```
1 //create a component to add to the frame
2 final JTextArea comp = new JTextArea();
3 JButton btn = new JButton("click");
4
```


16 Technical Key Areas

Step 3: Add your components to your display area and arrange or layout your components using the LayoutManagers. You can use the standard layout managers like FlowLayout, BorderLayout, etc. Complex layouts can be simplified by using nested containers for example having JPanels within JPanels and each JPanel can use its own LayoutManager. You can create components and add them to whichever JPanels you like and JPanels can be added to the JFrame's content pane.

```
1 // Add the component to the frame's content pane;
2 // by default, the content pane has a border layo
3 frame.getContentPane().add(comp, BorderLayout.CEN
4 frame.getContentPane().add(btn, BorderLayout.SOUT
```

Step 4: Attach listeners to your components. Interacting with a Component causes an Event to occur. To associate a user action with a component, attach a listener to it. Components send events and listeners listen for events. Different components may send different events, and require different listeners. The listeners are interfaces, not classes.

Step 5: Show the frame.

```
1 // set the frame size and Show the frame
2  int width = 300;
3  int height = 300;
4  frame.setSize(width, height);
5  frame.setVisible(true);
6
```

Note: For Applets, you need to write the necessary HTML code.

Q3. Explain the Swing Action architecture?
A3. The **Swing Action architecture** is used to implement shared behavior between two or more user interface components. For example, the menu items and the tool bar buttons will be performing the same action no matter which one is clicked. Another distinct advantage of using actions is that when an action is disabled then all the components, which use the Action, become disabled.

⊞ Judging Experience I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
⊞ Can you write code? (
⊞ ♦ Complete the given
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A

Design pattern: The javax.swing.Action interface extends the ActionListener interface and is an abstraction of a command that does not have an explicit UI component bound to it. The Action architecture is an implementation of **command design pattern**. This is a powerful design pattern because it allows the separation of controller logic of an application from its visual representation. This allows the application to be easily configured to use different UI elements without having to re-write the control or call-back logic.

Defining action classes:

```
1  class FileAction extends AbstractAction {
2      //Constructor
3     FileAction(String name) {
4           super(name);
5     }
6
7     public void actionPerformed(ActionEvent ae){
8          //add action logic here
9    }
10 }
11
```

To add an action to a menu bar:

```
1 JMenu fileMenu =  new JMenu(“File”);
2 FileAction newAction = new FileAction(“New”);
3 JMenuItem item =  fileMenu.add(newAction);
4 item.setAccelarator(KeyStroke.getKeyStroke(‘N’, E
5
```

To add action to a toolbar

```
1 private JToolBar toolbar = new JToolBar();
2 toolbar.add(newAction);
3
```

So, an action object is a listener as well as an action.

Q4. How does Swing painting happen? How will you improve the painting performance?
A4. If you want to create your own custom painting code or troubleshoot your Swing components, then you need to understand the basic concept of Swing painting.

- Swing GUI painting starts with the highest component that needs to be repainted and works it way down the hierarchy of components. This painting process is coordinated by the AWT painting system, but Swing repaint manager and double-buffering code, which means an off-screen buffer [image] is used during drawing and then the resulting bits are copied onto the screen. The resulting image is smoother, less flicker and quicker than drawing directly on the screen.
- Swing components generally repaint themselves whenever necessary. For example when you invoke the setTextt() on a component etc. This happens behind the scenes using a callback mechanism by invoking the repaint() method. If a component's size or position needs to change then the call to revalidate() method precedes the call to repaint() method.
- Like event handling code, painting code executes on the event-dispatching thread (Refer Q62 in Java Section). So while an event is being handled, no painting will occur and similarly while painting is happening no events will take place.
- You can provide your own painting by overriding the paintComponent() method. This is one of 3 methods used by JComponents to paint themselves.

```
1  public class MyFramePainting extends JFrame {
2      public static void main(String[] args) {
3          JFrame frame = new JFrame("Frame Title")
4
5          MyPanel panel = new MyPanel();
6          panel.setOpaque(true);        //if opaque
7                                        //does not
8          panel.setBackground(Color.white);
9          panel.setLayout(new FlowLayout());
10
11         ...//add to contentPane, display logic e
12     }
13 }
```

```
1  public class MyPanel extends JPanel implements M
2
3      Color col = Color.blue;
4
5      public void paintComponent(Graphics gr){
```

```
6           super.paintComponent(gr);
7
8           gr.setColor(col);
9           gr.drawLine(5,5, 200,200);
10      }
11
12      public MyPanel(){
13          addMouseListener(this); //i.e the Panel
14      }
15
16      public void mouseClicked(MouseEvent ev){
17          col = Color.red;
18          repaint(); //invokes paintComponent(). N
19      }
20
21      ...//other mouse events like onMousePressed
22  }
```

By default, the paintComponent() method paints the background if the component is opaque, then it performs any custom painting. The other two methods are paintBorder(Graphics g) and paintChildren(Graphics g), which tells to paint any border and paint any components contained by this component respectively. You should not invoke or override these two methods.

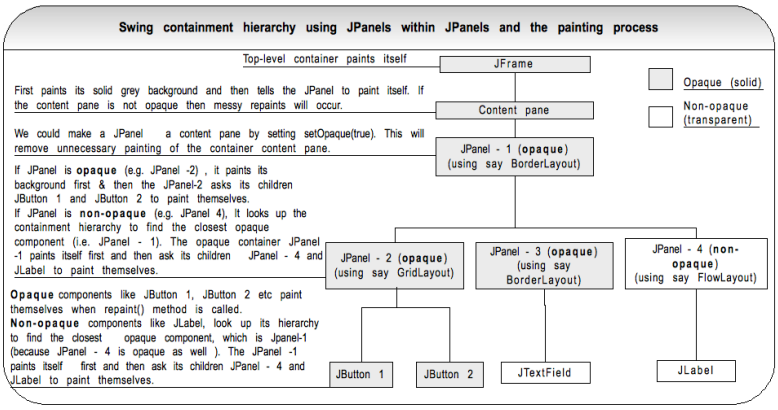
Q5. How will you improve the painting performance?
A5. On components with complex output, the repaint() method should be invoked with arguments which define only the clip rectangle that needs updating (rectangle origin is on top left corner). Note: No paintXXXX() methods (including paint() method) should not be explicitly invoked. Only repaint() method can be explicitly invoked (which implicitly calls paintComponent() method) and only paintComponent() should be overridden if required.

```
1  public void mouseClicked(MouseEvent ev){
2      col = Color.red;
3      repaint(0,0,50,50); //invokes paintComponent
4   }
5
```

You should never turn off double buffering for any Swing components.

The Swing painting efficiency can be optimized by the following two properties:

opaque: If the opaque (i.e. solid) property is set to true with myComponent.setOpaque(true) then the Swing painting system does not have to waste time trying to paint behind the component hence improves performance.



Swing painting

optimizedDrawingEnabled: This is a read only property (isOptimizedDrawingEnabled()) in JComponent, so the only way components can change the default value is to subclass and override this method to return the desired value. It's always possible that a non-ancestor component in the containment tree could overlap your component. In such a case the repainting of a single component within a complex hierarchy could require a lot of tree traversal to ensure 'correct' painting occurs.

true: The component indicates that none of its immediate children overlap.
false: The component makes no guarantees about whether or not its immediate children overlap

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Processing large files efficiently in Java – part 2

Q6 – Q11 Swing interview questions and answers ›

Posted in member-paid, Swing & AWT

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer

 ↑