Oral Questions for Java Experiments

Here is a comprehensive set of viva-voce style questions and answers based on the six Java programs you provided.

Experiment 1: TreeSetDemo

Concept: `java.util.TreeSet` (Sorted Set), Set operations.

Q1: What is a `TreeSet` in Java? A: A `TreeSet` is a class in the Java Collections Framework that implements the `Set` interface. It stores elements in a sorted order (natural ascending order by default) and does not allow duplicate elements.

Q2: What package do you need to import to use `TreeSet` ? A: `java.util.TreeSet` .

Q3: How does `TreeSet` store its elements to keep them sorted? A: It uses a balanced binary search tree, specifically a Red-Black Tree, internally (by using a `TreeMap`). This ensures that elements are always in sorted order and provides efficient `O(log n)` time for add, remove, and contains operations.

Q4: In the code, `tree1.add("zABCD");` is called twice. Will the set contain this string twice? A: No. `TreeSet` implements the `Set` interface, which does not allow duplicate elements. The first `add` will add the string and return `true` . The second `add` will see the element already exists, do nothing, and return `false` .

Q5: What will be the output of the first `System.out.println` for `tree1` ? A: `tree1 elements: [1, A, B, C, D, zABCD]` .

Q6: Why does "zABCD" come *after* "D" in the sorted order? A: Because sorting is lexicographical (dictionary-style) based on Unicode/ASCII values. Uppercase letters ('A' through 'Z') have lower values than lowercase letters ('a' through 'z'). So, "D" comes before "zABCD".

Q7: Why does "1" come *before* "A"? A: For the same reason. In ASCII/Unicode, the characters for digits ('0' through '9') come before the characters for uppercase letters.

Q8: What interface must objects implement to be stored in a `TreeSet` ? A: They must implement the `Comparable` interface (to define their "natural order"). Alternatively, you can pass a `Comparator` object to the `TreeSet` constructor to define a custom sorting order.

Q9: Why do `String` , `Integer` , and `Float` work in a `TreeSet` without any extra code? A: Because these classes (and all wrapper classes) already implement the `Comparable` interface by default.

Q10: What does the `tree1.remove("B")` method return? A: It returns a `boolean` . It will return `true` because "B" was present in the set and was successfully removed. If "B" was not in the set, it would return `false` .

Q11: What is the purpose of the line `TreeSet<String> union = new TreeSet<>(tree1);` ? A: This line creates a *new* `TreeSet` called `union` and initializes it with all the elements that are *currently* in `tree1` . This is the first step for performing the union operation.

Q12: What does the `union.addAll(tree2);` **method do? A:** This performs the set **union** operation. It adds all elements from `tree2` into the `union` set. Any duplicate elements from `tree2` that are already in `union` are ignored.

Q13: What does the `intersection.retainAll(tree2);` **method do? A:** This performs the set **intersection** operation. It *removes* any elements from the `intersection` set that are *not* also present in `tree2`. Only the common elements are kept.

Q14: What does the `difference.removeAll(tree2);` **method do? A:** This performs the set **difference** operation (specifically `tree1 - tree2`). It removes any elements from the `difference` set that are also present in `tree2`.

Q15: After `tree1.remove("B")` **, what are the contents of** `tree1` **and** `tree2` **? A:**

- `tree1` : `[1, A, C, D, zABCD]`
- `tree2` : `[1, 2, 3, 4, A]`

Q16: Based on Q15, what will be the output for the Intersection? A: `Intersection of tree1 & tree2: [1, A]` (These are the only two elements in both sets).

Q17: Based on Q15, what will be the output for the Difference (`tree1 - tree2` **)? A:** `Difference : tree1 - tree2 = [C, D, zABCD]` (These are the elements in `tree1` but not in `tree2`).

Q18: What is the main difference between a `TreeSet` **and a** `HashSet` **? A:** Both are Sets and don't allow duplicates. The main difference is:

- `TreeSet` : Is sorted. Operations are `O(log n)`.
- `HashSet` : Is unsorted. Operations are `O(1)` on average.

Q19: What would happen if you tried to add `null` **to this** `TreeSet` **? A:** It would throw a `NullPointerException` at runtime. `TreeSet` cannot store `null` because it needs to call the `compareTo` method on every element to maintain order, and you can't call a method on `null`.

Q20: What is the purpose of the `Scanner` **in this program? A:** To get input from the user in the console. It's used to read the number of elements `n` and then read `n` float values from the user to populate the `t2` (Float) `TreeSet`.

Q21: What will the `t1` **(Integer)** `TreeSet` **contain? A:** The loop adds `i * 5 + i * 2`, which is `i * 7`. For `i` from 0 to 4, it will add (0, 7, 14, 21, 28). The set will be `[0, 7, 14, 21, 28]`.

Experiment 2: sampleLinkedList

Concept: `java.util.LinkedList` (Doubly-Linked List), List and Deque operations.

Q1: What is a `LinkedList` **in Java? A:** It's a class in the Java Collections Framework that implements the `List` and `Deque` (Double-Ended Queue) interfaces. It stores elements as a sequence of nodes, where each node points to the next and previous node (a doubly-linked list).

Q2: What is the main structural difference between a `LinkedList` **and an** `ArrayList` **? A:**

- `ArrayList` : Uses a dynamic array internally. Fast for random access (get/set by index).
- `LinkedList` : Uses nodes with pointers. Fast for adding/removing elements from the beginning or end.

Q3: In the code, `l1.add(num);` is called inside the loop. Where is the new element added? A: By default, `add(element)` adds the element to the *end* of the list.

Q4: Does a `LinkedList` allow duplicate elements? A: Yes. Since it's a `List`, it allows duplicates and maintains the order in which elements were inserted.

Q5: Does a `LinkedList` allow `null` elements? A: Yes, it does.

Q6: What does the `l1.getLast()` method do? Does it change the list? A: It *retrieves* (returns) the last element in the list. It does *not* remove the element or change the list.

Q7: What does the `l1.getFirst()` method do? Does it change the list? A: It *retrieves* (returns) the first element in the list. It also does *not* remove the element.

Q8: What does the `l1.removeLast()` method do? A: It *removes* and *returns* the last element from the list. The size of the list is reduced by one.

Q9: What does the `l1.removeFirst()` method do? A: It *removes* and *returns* the first element from the list.

Q10: What does the `l1.remove()` method (with no arguments) do? A: It retrieves and removes the *head* (the first element) of the list. It is equivalent to `removeFirst()`.

Q11: Let's trace the code. Assume the user enters: `10, 20, 30, 40, 50`. What is the list `l1` after the loop? A: `[10, 20, 30, 40, 50]`

Q12: (Continuing from Q11) What will `System.out.println(l1.getLast());` print? A: `50`

Q13: (Continuing from Q11) What will `System.out.println(l1.getFirst());` print? A: `10`

Q14: (Continuing from Q11) What will `System.out.println(l1.removeLast());` print? And what is the list `l1` *after* this line? A: It will print `50`. The list `l1` will become `[10, 20, 30, 40]`.

Q15: (Continuing from Q14) What will `System.out.println(l1.removeFirst());` print? And what is the list `l1` *after* this line? A: It will print `10`. The list `l1` will become `[20, 30, 40]`.

Q16: (Continuing from Q15) What will `System.out.println(l1.remove());` print? And what is the list `l1` *after* this line? A: It will print `20`. The list `l1` will become `[30, 40]`.

Q17: When would you choose to use a `LinkedList` over an `ArrayList`? A: You would use a `LinkedList` when you have a large number of insertions and deletions, especially at the *beginning* or *middle* of the list, and you do not need frequent random access by index.

Q18: When would you choose an `ArrayList` over a `LinkedList`? A: You would use an `ArrayList` when you need fast random access (using `get(index)`) and most of your operations are reading or adding/removing from the *end* of the list.

Q19: Since `LinkedList` implements the `Deque` interface, how can you use it as a Stack? A: You can use the `push()` (add to front) and `pop()` (remove from front) methods.

Q20: How can you use `LinkedList` as a Queue? A: You can use `add()` (add to end) and `poll()` (remove from front).

Q21: The code imports `java.util.Arrays`. Is this import necessary for this program? A: No. The `Arrays` class is not used in this specific program. It can be removed without any effect.

Experiment 3: MouseDemo

Concept: AWT (Abstract Window Toolkit), Event Handling, `MouseListener` Interface.

Q1: What is AWT? A: AWT stands for Abstract Window Toolkit. It's Java's original, platform-dependent library for creating Graphical User Interfaces (GUIs).

Q2: What is the purpose of this program? A: To create a window that listens for mouse events (clicks, presses, entering, exiting) and displays a text `Label` describing the most recent mouse action.

Q3: What does `extends Frame` mean? A: It means the `MouseDemo` class *is a* `Frame`. It inherits all the properties and methods of the `Frame` class, such as `setSize`, `setVisible`, and `add`.

Q4: What does `implements MouseListener` mean? A: It's a contract. It means the `MouseDemo` class *must* provide implementations for all five methods defined in the `MouseListener` interface.

Q5: What are the five methods of the `MouseListener` interface? A: `mouseClicked`, `mousePressed`, `mouseReleased`, `mouseEntered`, and `mouseExited`.

Q6: What happens if you implement `MouseListener` but forget to define one of the methods, for example `mouseReleased`? A: The code will not compile. The compiler will complain that `MouseDemo` is not abstract and does not override the abstract method `mouseReleased`.

Q7: What is the difference between `mouseClicked` and `mousePressed`? A:

- `mousePressed`: Fires immediately when the mouse button is pushed down.
- `mouseClicked`: Fires *after* a `mousePressed` and `mouseReleased` happen in quick succession at the same location.

Q8: What does the `addMouseListener(this);` line do? A: It registers the current object (the `MouseDemo` frame, referred to by `this`) as a listener for mouse events. The `Frame` is both the *source* of the events and the *listener* that handles them.

Q9: What does `setLayout(null);` do? A: It disables the default layout manager for the `Frame`. This allows you to manually set the exact pixel position and size of components using `setBounds()`.

Q10: Is `setLayout(null)` considered good practice in modern Java GUI development? A: No, it is generally discouraged. Using layout managers (like `FlowLayout`, `BorderLayout`, `GridLayout`) is preferred because they create responsive UIs that look good even when the window is resized.

Q11: What is the `l.setBounds(20, 50, 300, 20);` line doing? A: It's setting the position and size of the `Label` component. It will be at (x=20, y=50) and will have a width of 300 pixels and a height of 20 pixels.

Q12: What is a `WindowAdapter`? A: It's a convenience class that provides empty implementations for all methods in the `WindowListener` interface. This allows you to override *only* the methods you care about.

Q13: Why is the `WindowAdapter` used here? A: It's used to handle the window-closing event. By overriding just `windowClosing(WindowEvent e)`, we can call `dispose()` to close the window and shut down the program when the user clicks the 'X' button.

Q14: What would happen if you removed the entire `addWindowListener` block? A: The window would appear, but clicking the 'X' button would *not* stop the Java program. The window would close, but the application would still be running in the background.

Q1What is the `MouseEvent e` parameter that is passed to the listener methods? A: It's an event object that contains detailed information about the mouse event, such as its source, and in the case of `mouseClicked`, the X and Y coordinates of the click (`e.getX()` and `e.getY()`).

Q16: When does the `mouseEntered` event fire? A: It fires when the mouse cursor moves *into* the boundaries of the component the listener is registered on (in this case, the `Frame`).

Q17: When does the `mouseExited` event fire? A: It fires when the mouse cursor moves *out of* the component's boundaries.

Q18: What is the difference between AWT and Swing? A: AWT components are "heavyweight," meaning they rely on the native operating system's UI components. Swing components (in `javax.swing`) are "lightweight," meaning they are drawn entirely in Java and are platform-independent.

Q19: What is the `super("Mouse Event Demo");` call? A: It calls the constructor of the parent class (`Frame`) to set the title of the window.

Q20: How could you avoid implementing all five `MouseListener` methods if you only cared about `mouseClicked` ? A: You could use a `MouseAdapter` class. You would `extend MouseAdapter` (instead of `implements MouseListener`) and then you only need to override the `mouseClicked` method.

Experiment 4: KeyboardDemo

Concept: AWT (Abstract Window Toolkit), Event Handling, `KeyListener` Interface.

Q1: What is the purpose of this program? A: To demonstrate keyboard event handling. It has a `TextField` that listens for key presses and updates a `Label` to describe the key event.

Q2: What interface is implemented for keyboard events? A: The `KeyListener` interface.

Q3: What are the three methods of the `KeyListener` interface? A: `keyPressed` , `keyReleased` , and `keyTyped` .

Q4: What is the difference between `keyPressed` and `keyTyped` ? A:

- `keyPressed` : Fires for *any* physical key press, including non-character keys like Shift, Ctrl, F1, or the arrow keys.
- `keyTyped` : Fires only when a key press results in a valid Unicode character being generated. It won't fire for keys like Shift or F1.

Q5: What is the difference between `keyPressed` and `keyReleased` ? A: `keyPressed` fires when the key is pushed down. `keyReleased` fires when the key is let go.

Q6: In this code, which component is the `KeyListener` added to? A: The `TextField` (`tf.addKeyListener(this);`).

Q7: Why is the listener added to the `TextField` and not the `Frame`? A: Because a `Frame` doesn't normally "have focus" in a way that lets it receive key events. A `TextField` is *designed* to receive keyboard input, so it must have "focus" for the `KeyListener` to work.

Q8: What is "focus" in a GUI? A: It means the component is currently active and ready to receive input (like keyboard strokes). You can usually see it as a blinking cursor.

Q9: In `keyPressed`, what does `e.getKeyCode()` return? A: It returns an integer "virtual key code" that represents the *physical* key on the keyboard, regardless of whether Shift was pressed (e.g., `KeyEvent.VK_A` for the 'A' key).

Q10: What does `KeyEvent.getKeyText(e.getKeyCode())` do? A: It converts that integer key code into a human-readable string, like "A", "Enter", or "F1".

Q11: In `keyTyped`, what does `e.getKeyChar()` return? A: It returns the actual `char` value of the key that was typed. For example, if you press 'a', it returns 'a'. If you press Shift+'a', it returns 'A'.

Q12: If you press and release the 'Shift' key, what events will fire? A: `keyPressed` (for Shift) and `keyReleased` (for Shift). No `keyTyped` event will fire, because 'Shift' itself is not a character.

Q13: If you press and release the 'A' key (without Shift), what events will fire, and in what order? A:

1. `keyPressed` (for 'A')
2. `keyTyped` (for 'a')
3. `keyReleased` (for 'A')

Q14: If you press and release the '5' key on the number pad, what will `keyPressed` report? A: `Key Pressed: Numpad-5`.

Q15: If you press and release the '5' key on the top row, what will `keyPressed` report? A: `Key Pressed: 5`.

Q16: The `keyReleased` method just sets the label to "Key Released". How could you make it show *which* key was released? A: You would use the same code as in `keyPressed`: `l.setText("Key Released: " + KeyEvent.getKeyText(e.getKeyCode()));`

Q17: What is a `TextField`? A: An AWT component that allows the user to enter a single line of text.

Q18: What does the `super("Keyboard Event Demo");` call do? A: It calls the constructor of the parent `Frame` class to set the window's title.

Q19: This program also uses `setLayout(null)`. Why? A: To manually position the `Label` and `TextField` using `setBounds()`.

Q20: Is there a `KeyAdapter` class, similar to `WindowAdapter`? A: Yes. If you only wanted to handle `keyPressed`, you could `extend KeyAdapter` and only override that one method, instead of `implements KeyListener` and providing empty implementations for the other two.

Experiment 5: ColorTextDemo

Concept: AWT Graphics, `paint()` method, `Graphics`, `Font`, and `Color` classes.

Q1: What is the purpose of this program? A: To create a window with a black background and draw the string "AllTheBest" five times, each in a different color, at different vertical positions.

Q2: How is this program different from the Mouse and Keyboard demos? A: This program doesn't have interactive components like `Label` or `TextField`. It does custom drawing directly onto the `Frame` using the `paint()` method.

Q3: What is the `paint(Graphics g)` method? A: It's a method from the `Component` class that the AWT system calls automatically whenever the window needs to be drawn. This happens when the window first appears, when it's resized, or when it's uncovered by another window.

Q4: What is the `Graphics g` parameter? A: It's the "graphics context" for the component. It's an object that provides all the drawing tools, like methods to draw strings, shapes, set colors, and set fonts.

Q5: Should you ever call the `paint()` method directly yourself? A: No. If you need to force a component to redraw, you should call the `repaint()` method. `repaint()` tells the AWT system to schedule a call to `paint()` as soon as possible.

Q6: How is the background color set to black? A: By calling `setBackground(Color.BLACK);` in the constructor.

Q7: What does `g.setFont(new Font("Arial", Font.BOLD, 20));` do? A: It sets the current font for all subsequent text-drawing operations on this `Graphics` object. The font will be Arial, bold, and 20 points in size.

Q8: What does `g.setColor(Color.RED);` do? A: It sets the current drawing color for all subsequent drawing operations (text, shapes, etc.). The color will be red until it is changed again.

Q9: What does `g.drawString("AllTheBest", 50, 70);` do? A: It draws the string "AllTheBest" at the (x, y) coordinates (50, 70).

Q10: In `g.drawString(string, x, y)`, what do the `x` and `y` coordinates represent? A: `x` is the horizontal position (pixels from the left edge). `y` is the vertical position (pixels from the top edge). Importantly, `y` represents the **baseline** of the text, not the top of the text.

Q11: What are `Color.RED`, `Color.BLUE`, etc.? A: They are `public static final` constants defined in the `java.awt.Color` class, providing easy access to common colors.

Q12: How would you create a custom color, for example, a light gray? A: You would create a new `Color` object using its RGB (Red, Green, Blue) values: `Color lightGray = new Color(200, 200, 200);` `g.setColor(lightGray);`

Q13: What happens if you resize the window? A: The `paint()` method is called again, and all five lines of text are redrawn at their fixed coordinates.

Q14: What is the font style `Font.BOLD`? What are the other basic styles? A: It's an integer constant. The other common styles are `Font.ITALIC` and `Font.PLAIN` (which is the default).

Q15: How could you draw text that is both bold and italic? A: You would combine the styles using the bitwise OR operator: `g.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 20));`

Q16: Why is `setVisible(true)` called in the constructor? A: To make the `Frame` window actually appear on the screen.

Q17: Why doesn't this program need any `MouseListener` or `KeyListener` ? A: Because it's not interactive. It only displays information and doesn't respond to user input (other than the standard window-closing event).

Q18: What packages are needed for this program? A: `java.awt.*` (for `Frame` , `Graphics` , `Font` , `Color`) and `java.awt.event.*` (for the `WindowAdapter` and `WindowEvent`).

Q19: Where is the (0, 0) coordinate on the `Frame` ? A: The top-left corner of the `Frame` 's drawable area (just inside the title bar and window borders).

Q20: What's the last line of text drawn and what is its color? A: `g.drawString("AllTheBest", 50, 190);` and its color is `Color.MAGENTA` .

Experiment 6: NumberDemo

Concept: Swing (javax.swing), Event Handling, `ActionListener` , `NumberFormatException` .

Q1: What is the main difference between this program and the AWT demos? A: This program uses **Swing** (`javax.swing.*`), which is Java's modern, lightweight, platform-independent GUI toolkit. The AWT demos used `java.awt.*` .

Q2: How can you tell it's a Swing program from the code? A: The class names start with "J": `JFrame` , `JTextField` , `JLabel` .

Q3: What is the purpose of this program? A: It's a simple application where a user can type a number, press Enter, and the program will display the previous number (number - 1) and the next number (number + 1).

Q4: This class `implements ActionListener` . What method must it define? A: The `actionPerformed(ActionEvent e)` method.

Q5: What is an `ActionEvent` ? A: It's an event that signifies a component-specific "action" has occurred. For a `JTextField` , the action is the user pressing the Enter key. For a `JButton` , it's the user clicking the button.

Q6: Which component is the `ActionListener` added to? A: The input text field: `tfInput.addActionListener(this);` .

Q7: This class does *not* `extend JFrame` . How does it create a window? A: It uses **composition** instead of inheritance. It creates an *instance* of `JFrame` and stores it in the `frame` variable. This is often considered a more flexible design.

Q8: What does `frame.setLayout(new FlowLayout());` do? A: It sets the layout manager for the `JFrame` to `FlowLayout` . This manager arranges components one after another in a row, and "wraps" to the next row if the window isn't wide enough.

Q9: What is the advantage of `FlowLayout` over `setLayout(null)` ? A: `FlowLayout` makes the UI responsive. If you resize the window, the components will re-arrange themselves neatly. `setLayout(null)` is rigid and looks bad when resized.

Q10: What does `tfPrevious.setEditable(false);` do? A: It makes the `JTextField` read-only. The user cannot type in it. This is good for fields that are for display-only.

Q11: What is the purpose of the `try...catch (NumberFormatException ex)` block? A: It's essential error handling. The line `Integer.parseInt(inputText)` will crash (throw a `NumberFormatException`) if the user types something that isn't a valid integer, like "hello" or "1.5".

Q12: What happens if the user types "abc" and presses Enter? A: The `try` block fails at `Integer.parseInt` . The program jumps to the `catch` block, which sets the `tfPrevious` and `tfNext` fields to display the text "Invalid".

Q13: What happens if the user types "20" and presses Enter? A: The `try` block succeeds. `number` becomes 20, `previous` becomes 19, `next` becomes 21. `tfPrevious` is set to "19" and `tfNext` is set to "21".

Q14: What does `String.valueOf(previous)` do? A: It converts the `int` variable `previous` (e.g., `19`) into its `String` representation (e.g., `"19"`). This is necessary because `setText()` requires a `String` .

Q15: What does `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` do? A: This is the modern Swing way to handle window closing. It tells the `JFrame` that when the user clicks the 'X' button, the entire Java application should terminate (exit the JVM).

Q16: How does Q15 compare to the `WindowAdapter` used in the AWT demos? A: `EXIT_ON_CLOSE` is much simpler and is the standard, preferred way to close a Swing application. The `WindowAdapter` is the more verbose AWT way.

Q17: What is `SwingUtilities.invokeLater(new Runnable() { ... });` ? A: This is a crucial piece of Swing code. It ensures that the GUI-creation code (`new NumberDemo()`) runs on the correct thread, called the **Event Dispatch Thread (EDT)**.

Q18: Why is the Event Dispatch Thread (EDT) important? A: Swing is *not* thread-safe. All code that creates, modifies, or interacts with Swing components *must* run on the EDT to prevent "race conditions" and visual glitches. `invokeLater` queues your `Runnable` to be executed on the EDT.

Q19: What is a `JLabel` ? A: A simple Swing component used to display a non-editable line of text or an icon. It's used here as a label for the text fields.

Q20: What is the `main` method in this file? A: It's the entry point for the application. Its only job is to use `SwingUtilities.invokeLater` to safely create an instance of `NumberDemo` on the EDT.

Q21: If you had two `JTextFields` and one `ActionListener` , how would you know which text field the user pressed Enter in? A: Inside `actionPerformed` , you would use the `ActionEvent` parameter `e` : `if (e.getSource() == tfInput) { ... } else if (e.getSource() ==`