Tutorial of Swing

> source code are based on lab materials of "sustech-teaching group"
>
> Document designed by ZHU Yueming in 2023. May. 6th

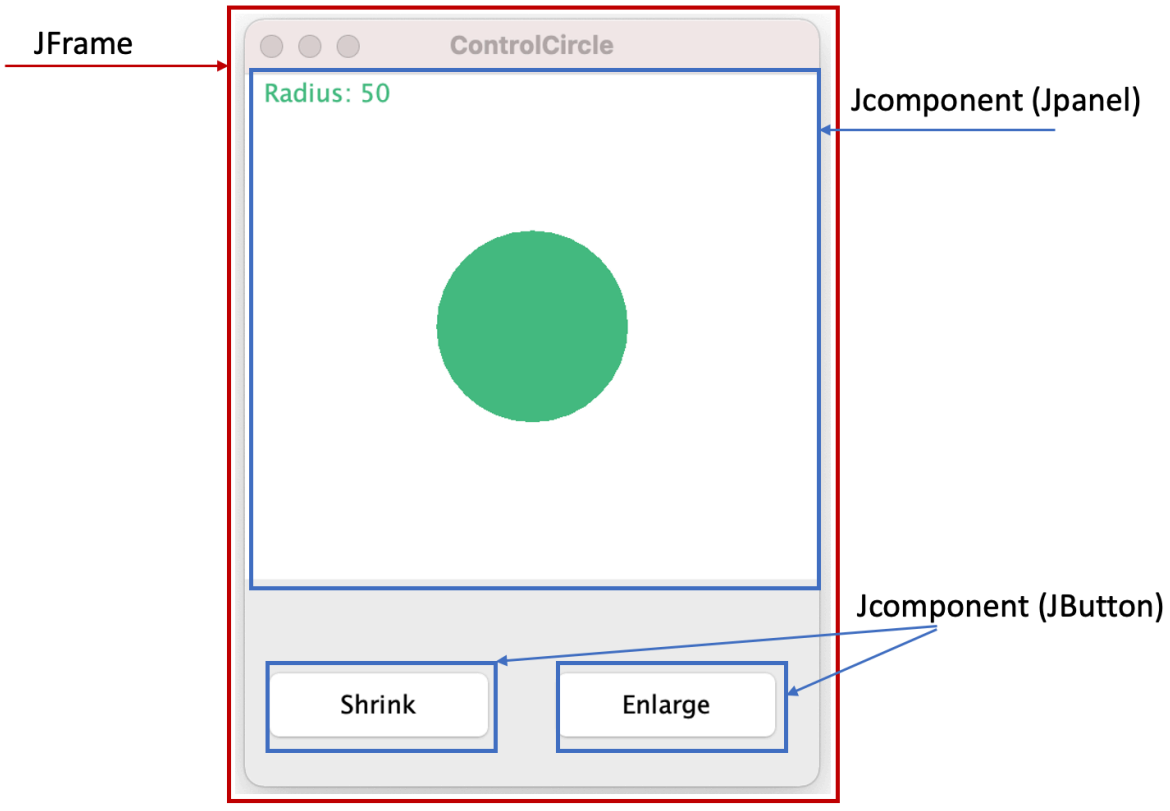
Objective

- Understand JFrame and JComponent
- Understand x and y coordinate in swing
- Can use the painting mechanism of swing
- Can use the mouse event in swing
- Understand addActionListener in swing

Introduction

1. JFrame and JComponent

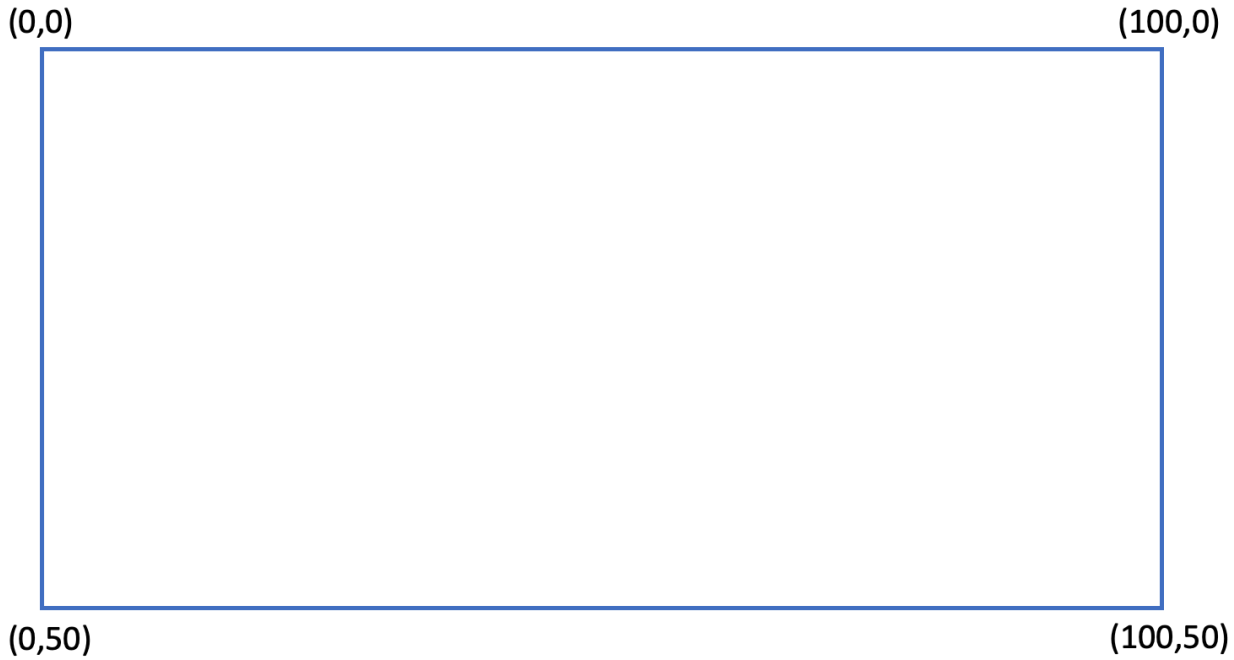
JFrame represents the form of the desktop application, and the components in the form are called JComponent.



2. x and y coordinate

x and y coordinates

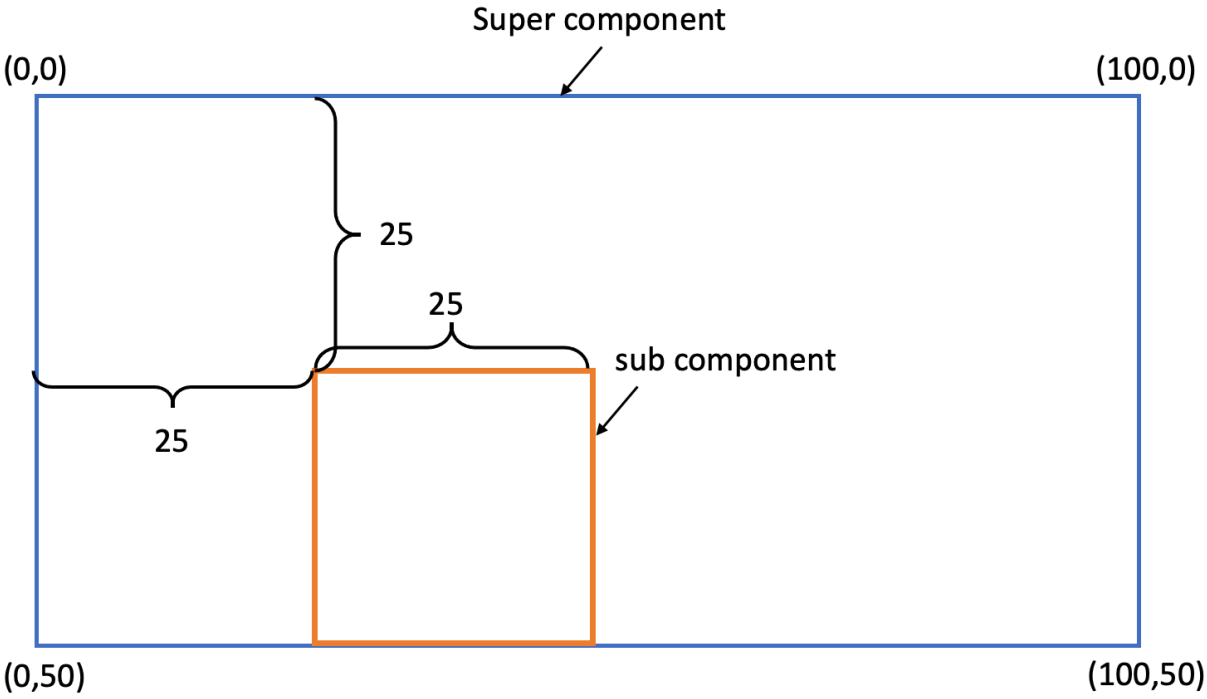
In Swing, the origin of x and y coordinates is in the upper left corner. For example, if the width and height are 100,50 respectively, the x and y coordinate of the four corner should be



location

In Swing, the location of current component represents the its upper left corner on the x and y coordinate of its super component.

For example:



The location of sub component in super component is: (25,25)

If we want the components to be displayed on it superComponent normally, we need to add the components to the super component and place them in a suitable location.

For example:

```
superComponent.setSize(50,100);
subComponet.setSize(25,25);
subComponet.setLocation(25,25);
superComponent.add(subComponent);
```

3. painting mechanism

All subclasses of `JComponent` can override a method `paintComponent`, in which you can paint anything you want in current component.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
}
```

The method cannot be invoked directly by us, while you can invoke `repaint()` method to execute the `paintComponent` method.

For example: when we invoke the `shrink()` method, it repaint a smaller circle.

```
  /**
    * Enlarge the circle
```

```
     */
    public void shrink() {
        radius = (int) (radius * 0.9);
        this.repaint();
    }

    /**
     * when doing repaint() method, execute paintComponent method
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(this.color);
        g.drawString(String.format("Radius: %d",this.radius),10,15);
        g.fillOval(this.getWidth() / 2 - radius, this.getHeight() / 2 - radius,
2 * radius, 2 * radius);
    }
```

4. mouse event

If one component has been add mouse click event, when it being clicked by mouse, the method processMouseEvent can execute immediately.

Add following code in component, which means the mouse click event:

```
enableEvents(AWTEvent.MOUSE_EVENT_MASK);
```

Design following method in the same component above, which mean the action would be processed as soon as the component is clicked:

```
 @Override
    protected void processMouseEvent(MouseEvent e) {
        super.processMouseEvent(e);
        if (e.getID() == MouseEvent.MOUSE_PRESSED) {
            color = new Color(random.nextInt(255), random.nextInt(255),
random.nextInt(255));
            System.out.println(color);
            repaint();
        }
    }
```

5. Listener in button

The `jbtEnlarge` and `jbtShrink` are `JButton` components.

```
JButton jbtEnlarge = new JButton("Enlarge");
JButton jbtShrink = new JButton("Shrink");
//add click listener into button: jbtEnlarge
jbtEnlarge.addActionListener(l -> {
      canvas.enlarge();//when the button clicked, do enlarge method.
});
//add click listener into button: jbtShrink
jbtShrink.addActionListener(l -> {
      canvas.shrink();//when the button clicked, do shrink method.
});
```

All statements that you want to execute after clicking the button should be written in `{}`.

```
btn.addActionListener(l -> {
      //todo: writing all statements that you want to execute after clicking
this btn
});
```