Programming in Java – Part 14
Working with Swing components

Paolo Vercesi

ESTECO SpA

Agenda

Basics of multithreading

Working with Swing components

Writing custom Swing components

Basics of multithreading

Concurrency

“more than one task running simultaneously on a system”

Writing correct programs is hard;
writing correct concurrent programs is harder.

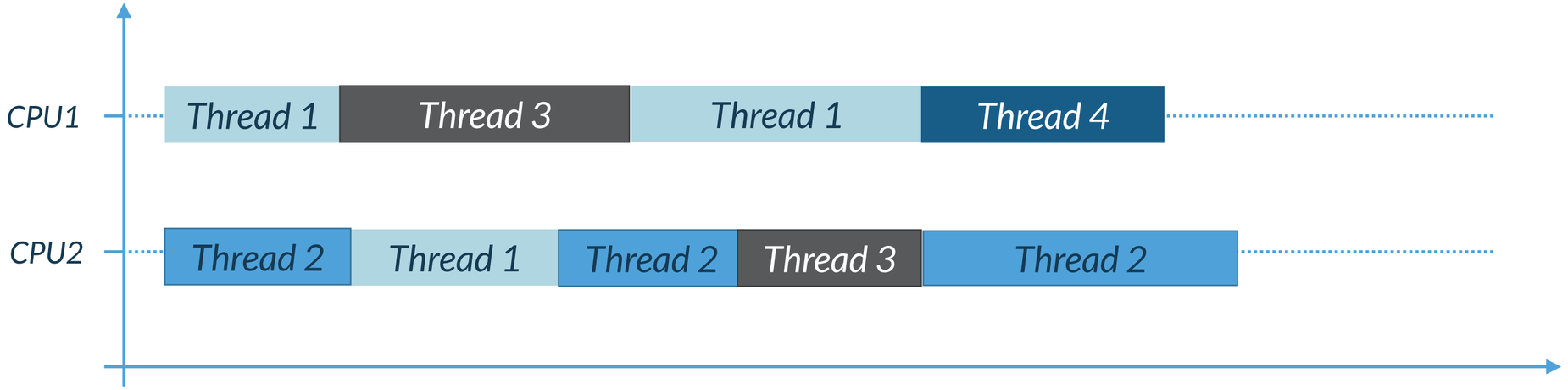
Processes and threads

A process is an executing program

A thread of execution, or simply a thread, is the smallest unit of execution to which a scheduler allocates a CPU

A Java process contains several concurrent threads executing in a shared memory environment

Java threads are scheduled (started, interrupted, and resumed) by the scheduler of the underlying operating system



The execution of a thread is assigned to a CPU until the scheduler decides to interrupt the thread execution to schedule another thread

Threads

A thread consists of a stack of calls, a program counter, and an id

In Java, threads are instances of the `java.lang.Thread` *class*

```
"RMI TCP Accept-0" #24 daemon prio=6 os_prio=0 cpu=0.00ms elapsed=325.06s tid=0x0000021167497000 nid=0x253c runnable
[0x0000004e2e0fe000]
   java.lang.Thread.State: RUNNABLE
        at java.net.PlainSocketImpl.accept0(java.base@11.0.15/Native Method)
        at java.net.PlainSocketImpl.socketAccept(java.base@11.0.15/PlainSocketImpl.java:159)
        at java.net.AbstractPlainSocketImpl.accept(java.base@11.0.15/AbstractPlainSocketImpl.java:474)
        at java.net.ServerSocket.implAccept(java.base@11.0.15/ServerSocket.java:565)
        at java.net.ServerSocket.accept(java.base@11.0.15/ServerSocket.java:533)
        at it.esteco.rmi.ssl.SslRMIServerSocketFactory$1.accept(SslRMIServerSocketFactory.java:24)
        at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.executeAcceptLoop(java.rmi@11.0.15/TCPTransport.java:394)
        at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.run(java.rmi@11.0.15/TCPTransport.java:366)
        at java.lang.Thread.run(java.base@11.0.15/Thread.java:829)
```

A thread is executing the code of a single method, namely the current method for that thread and the program counter contains the address of the instruction currently being executed

Starting a Thread

```
static void main() throws Exception {
  var thread = new Thread(new Runnable() {
    @Override
    public void run() {
      while (true) {
        System.out.println("Running");
        try {
           Thread.sleep(2000);
        } catch (Exception ex) {
           ex.printStackTrace();
        }
      }
    }
  });
  thread.start();
  System.out.println("End of main");
}
```

```
End of main
Running
Running
Running
…
```

The Java Virtual Machine allows an application to have multiple threads of execution running concurrently, regardless the number of processors

The `Thread` *class is used to create and start new threads of execution*

The Thread class

```
public class Thread implements Runnable {

    public Thread()

    public Thread(Runnable target)

    public Thread(String name)

    public Thread(Runnable target, String name)

    …
}
```

```
public interface Runnable {

    public abstract void run();
}
```

The `Runnable` object that a thread runs can be either the `Thread` itself or the target thread passed to the constructor

The `Thread` class implements an empty `run()` method

The two constructors without the `Runnable` target should be used by subclasses only

To use a thread, you must either pass a `Runnable` in the constructor or to override the `run()` method

Thread instance methods

```
public void start()
@Override
public void run()
public void interrupt()
public boolean isInterrupted()
public final boolean isAlive()
public final void setName(String name)
public final String getName()
public final void join(final long millis)
public final void join(long millis, int nanos) throws InterruptedException
public final void join() throws InterruptedException
public final void setDaemon(boolean on)
public final boolean isDaemon()
```

A thread does not return a value nor throw any exception

Some of Thread static methods

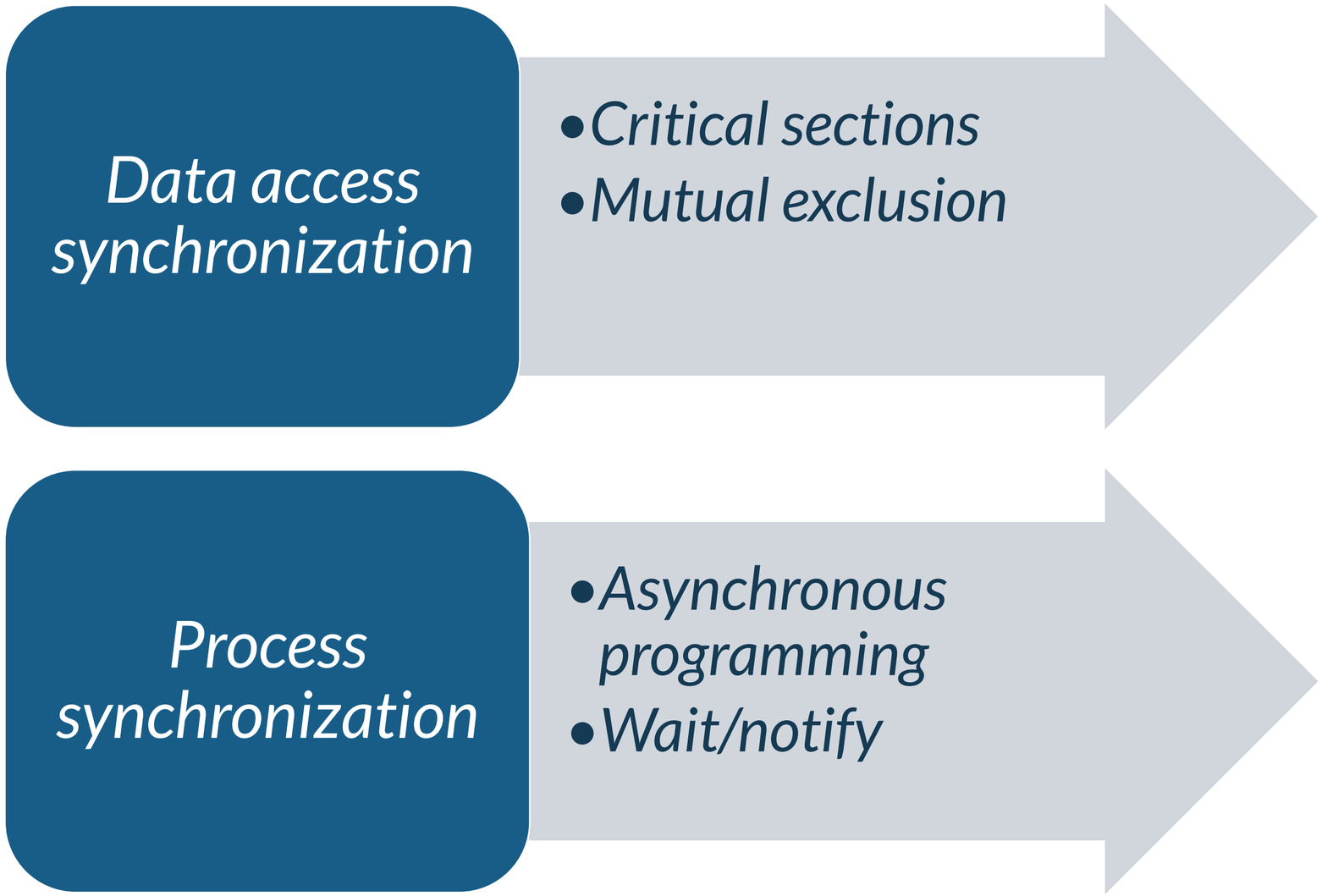
```
public static Thread currentThread()
public static void yield()
public static void sleep(long millis) throws InterruptedException
public static void sleep(long millis, int nanos) throws InterruptedException
public static boolean interrupted()
public static void dumpStack()
```

Waiting for a thread to finish

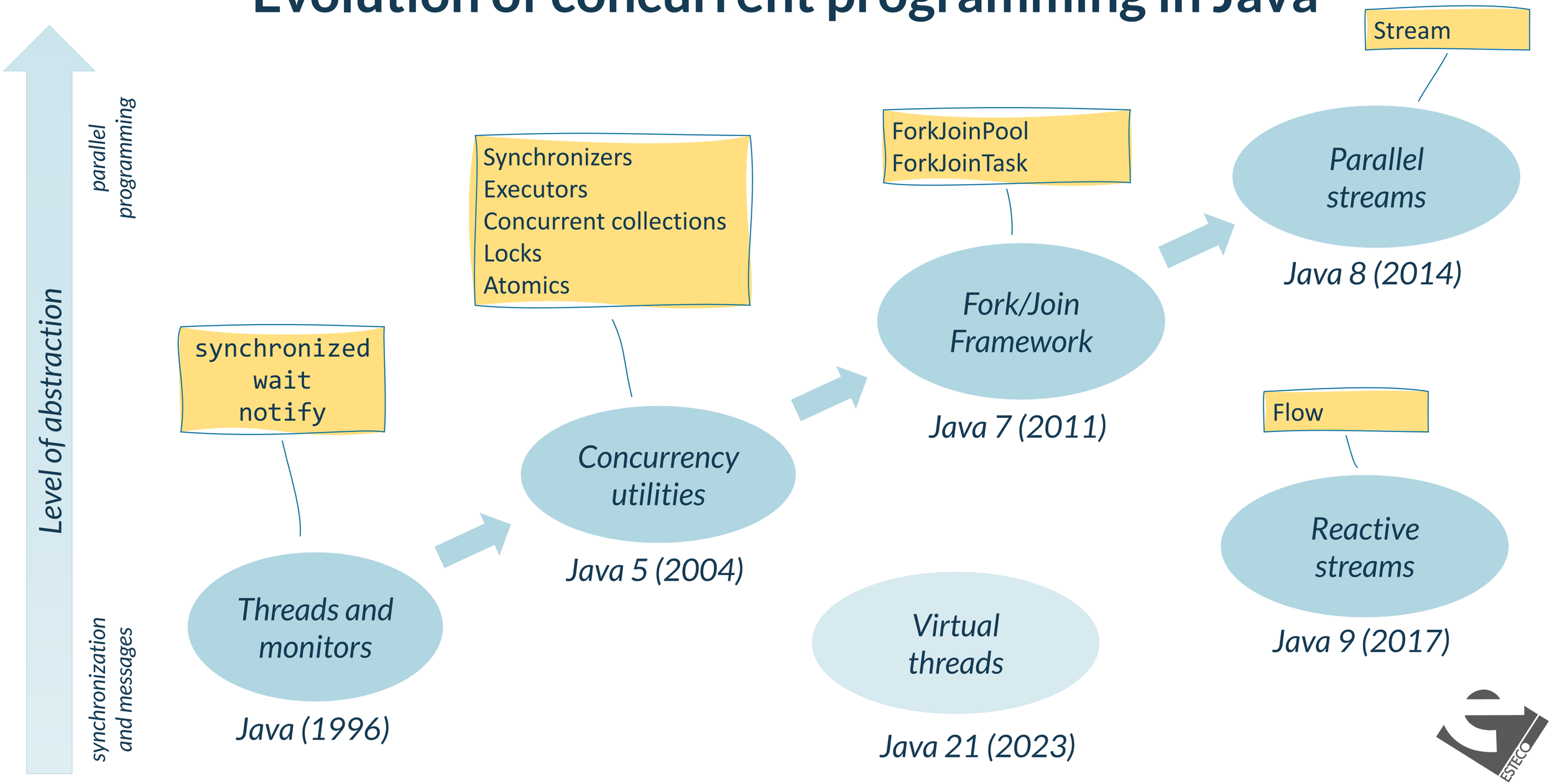
```
static void main() throws Exception {
    var thread = new Thread(new Runnable() {
        @Override
        public void run() {
            for (int i = 0; i < 5; i++) {
                System.out.println("Running");
                try {
                    Thread.sleep(1000);
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        }
    });
    thread.start();
    System.out.println("Start waiting for the thread to finish");
    thread.join();
    System.out.println("End of main");
}
```

```
Start waiting for the thread
to finish
Running
Running
Running
Running
Running
Running
End of main
```

Issues in concurrent programming



Evolution of concurrent programming in Java



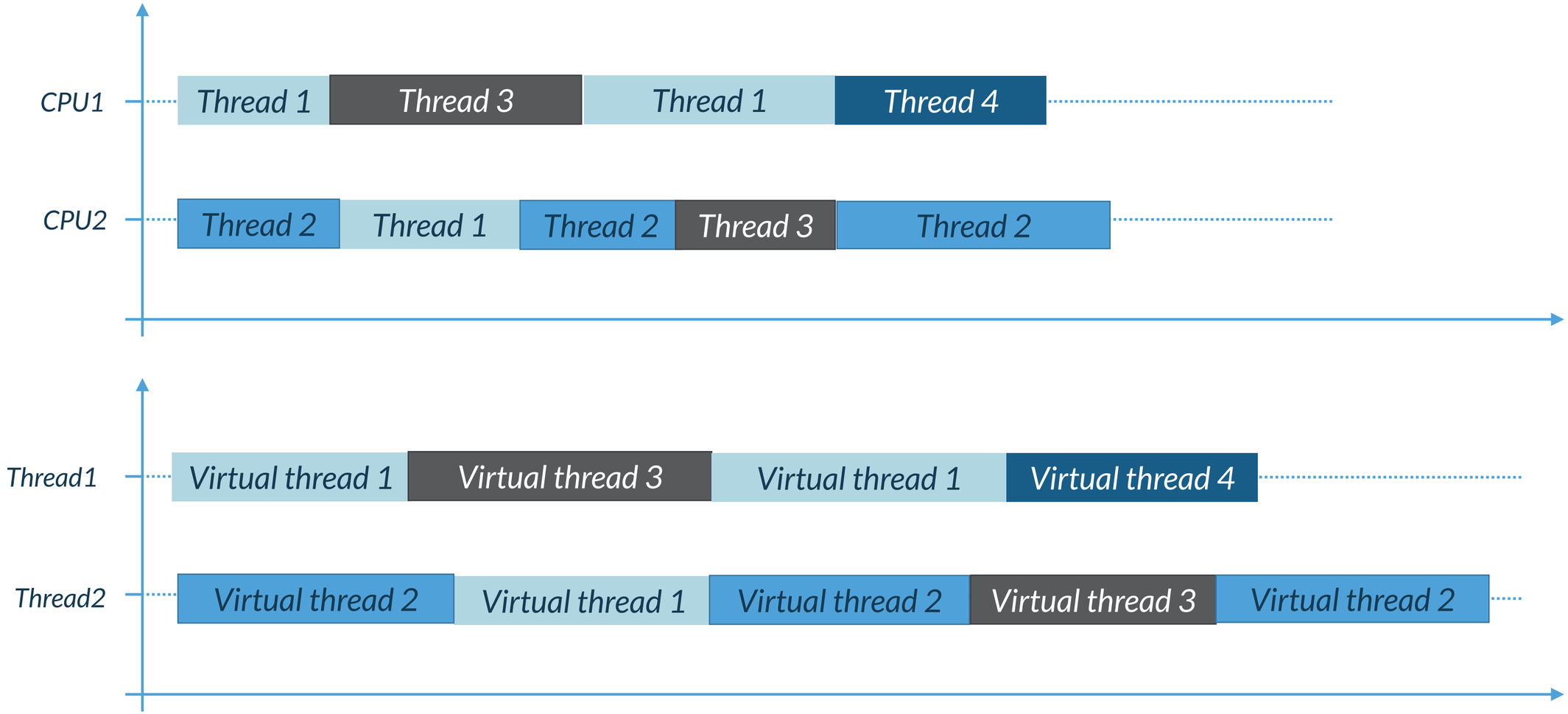
Virtual threads

- *So far, we have used the so-called platform threads*
- *A platform thread is a thin wrapper around an operating system thread*
- *A virtual thread is also an instance of* `java.lang.Thread`
  - *but it isn't tied to a specific operating system thread*
- *Virtual threads are stopped and resumed by the JVM not by the OS*
- *Virtual threads are not faster than platform threads*
- *So why were they introduced?*

High-throughput applications

- *Consider for example an application/web server*
- *It can concurrently run many database queries and/or many http connections*
- *Each of these queries or connections run on its own thread*
- *And each of these threads might be blocked waiting for the query results or the connection response*
- *Many operating system threads spend most of their lifetime waiting for some blocking I/O operation*
- *Operating system thread are considered a scarce resource*
- *So, it's a waste of resources to keep them blocked in a waiting state*
- *When a virtual threads is waiting, the associated operating system thread can be assigned to another virtual thread*

Platform vs virtual threads



https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html#GUID-DC4306FC-D6C1-4BCC-AECE-48C32C1A8DAA

Creating virtual threads

```
java.lang.Thread
public static Thread startVirtualThread(Runnable runnable)
public static Thread.Builder.OfPlatform ofPlatform()
public static Thread.Builder.OfVirtual ofVirtual()

java.lang.Thread.Builder
public Thread start(Runnable runnable)
public Thread unstarted(Runnable runnable)
```

Working with Swing components

Interactions with the GUI

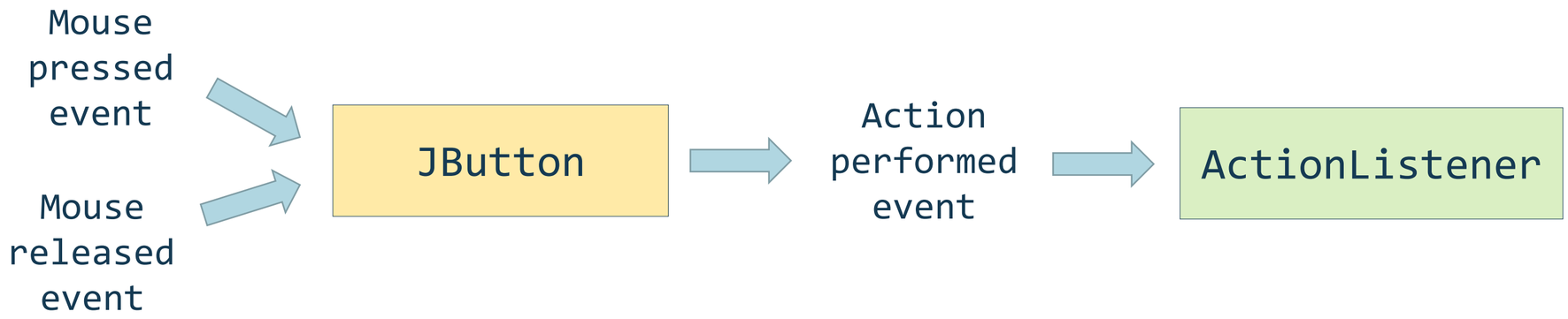
Swing components receive mouse and keyboard events from the window system, and they translate these events into events at the component level

In other words, Swing components fire events in response to user actions

Event processing happens in the event dispatch thread, as the name suggest

While processing events, it's always (thread) safe to invoke Swing methods from the same thread

From GUI events to component events



GUI events are dispatched to components. E.g., the Mouse pressed, and Mouse released events are dispatched to the `JButton`

Components translate GUI events into component events. E.g., the Mouse pressed and Mouse released events trigger an Action performed event

Registered listeners receive the component event. E.g., an `ActionListener` *registered to the* `JButton` *receives the Action performed event*

All these events are dispatched through the event dispatch thread

Swing is not thread-safe

Most Swing object methods are not thread-safe, invoking them from multiple threads risks thread interference or memory consistency errors

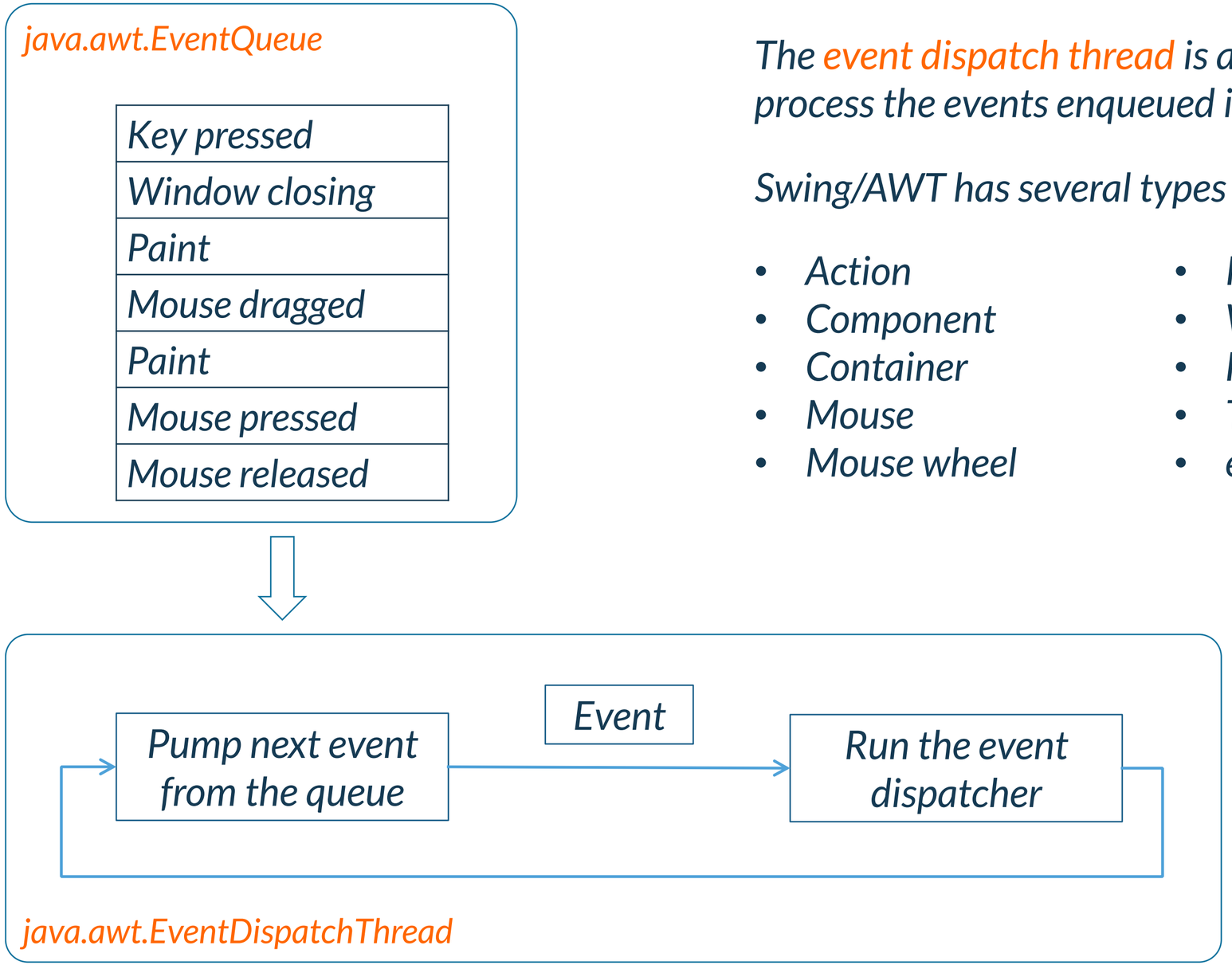
Some Swing component methods are labelled thread-safe in the API specification; these can be safely invoked from any thread. All other Swing component methods must be invoked from the event dispatch thread

Swing event handling code runs on a special thread known as the event dispatch thread (EDT) and most of the code that invokes Swing methods also runs on this thread

Programs that ignore this rule may seems to run correctly most of the times but are subject to unpredictable errors that are difficult to reproduce

https://docs.oracle.com/en/java/javase/21/docs/api/java.desktop/javax/swing/package-summary.html

The event queue & event dispatch thread



The event dispatch thread is a thread used to process the events enqueued in an event queue

Swing/AWT has several types of events

- *Action*
- *Component*
- *Container*
- *Mouse*
- *Mouse wheel*
- *Key*
- *Window*
- *Focus*
- *Text*
- *etc.*

Using the event dispatch thread

The code that handles Swing events is invoked from the event dispatch thread

If you need to determine whether your code is running on the event dispatch thread, invoke javax.swing.SwingUtilities.isEventDispatchThread

Tasks on the event dispatch thread must finish quickly; if they don't, unhandled events back up and the user interface becomes unresponsive

Longer tasks should run in background, i.e., without blocking the GUI by using a `SwingWorker`

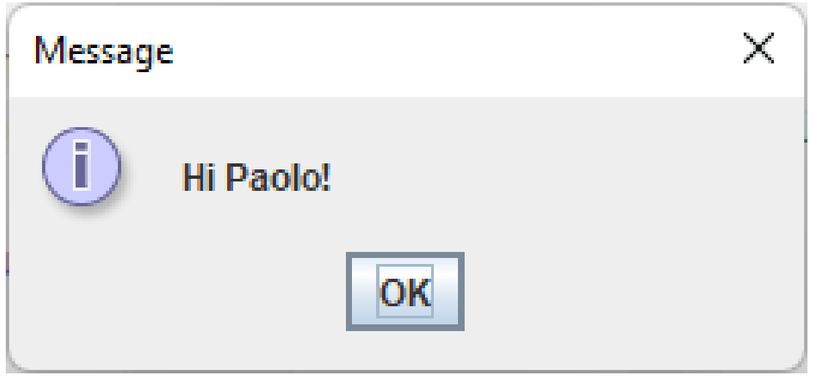
Swing “components”

- **containers**
  - *panel*
  - *scroll pane*
  - *split pane*
  - *tabbed pane*
  - *tool bar*
- **buttons**
  - *push button*
  - *check box*
  - *toggle button*
  - *radio button*
- **choosers**
  - *color chooser*
  - *file chooser*
- **combo box**
- **list**
- **menus**
  - *menu bar*
  - *popup menu*
  - *menu*
  - *menu item*
- **option pane**
- **panes**
  - *editor pane*
  - *text pane*
- **progress bar**
- **separator**
- **slider**
- **spinner**
- **table**
- **text components**
  - *text field*
  - *password field*
  - *text area*
  - *text pane*
- **tool tip**
- **tree**

JOptionPane

`JOptionPane` *can be used to inform the user about something or to ask for some input.*
The class has many public constructors and many static methods to show dialogs.

```
showMessageDialog()
showConfirmDialog()
showInputDialog()
showOptionDialog()
```



Parameters

- *parentComponent*
- *message*
- *messageType*
- *optionType*
- *options*
- *icon*
- *title*
- *initialvalue*

JOptionPaneDemo

OptionPaneDemo.java

```
import static javax.swing.JOptionPane.showConfirmDialog;
import static javax.swing.JOptionPane.showInputDialog;
import static javax.swing.JOptionPane.showMessageDialog;

public class JOptionPaneDemo {

    static void main() {
        SwingUtilities.invokeLater(JOptionPaneDemo::demo);
    }

    private static void demo() {
        String name = showInputDialog(null, "What's your name");
        int result = showConfirmDialog(null, "Your name is: " + name + "\n Is it right?");
        if (result == JOptionPane.OK_OPTION) {
            showMessageDialog(null, "Hi " + name + "!");
        } else {
            showMessageDialog(null, "Try again", "Incorrect name", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

SwingDemo

Swing demo – Setting up and showing the JFrame

```
JFrame frame = new JFrame("Swing Demo");
frame.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
Container cp = frame.getContentPane();
cp.setLayout(new BorderLayout());
JLabel label = new JLabel("Hello, World!");
label.setOpaque(true);
    …
cp.add(new JScrollPane(label), BorderLayout.CENTER);
cp.add(northPanel, BorderLayout.NORTH);
cp.add(southPanel, BorderLayout.SOUTH);
frame.setSize(600, 200);
frame.setVisible(true);
```

The JScrollPane

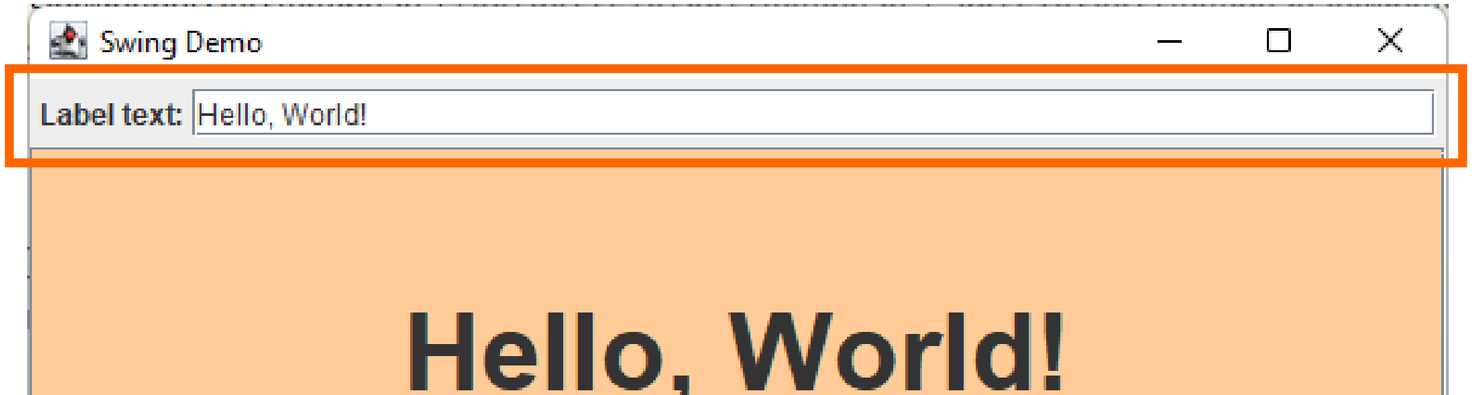
Hello, World!

The JScrollPane shows the component through a viewport

When the viewport is not wide enough, scrollbars are added to the view

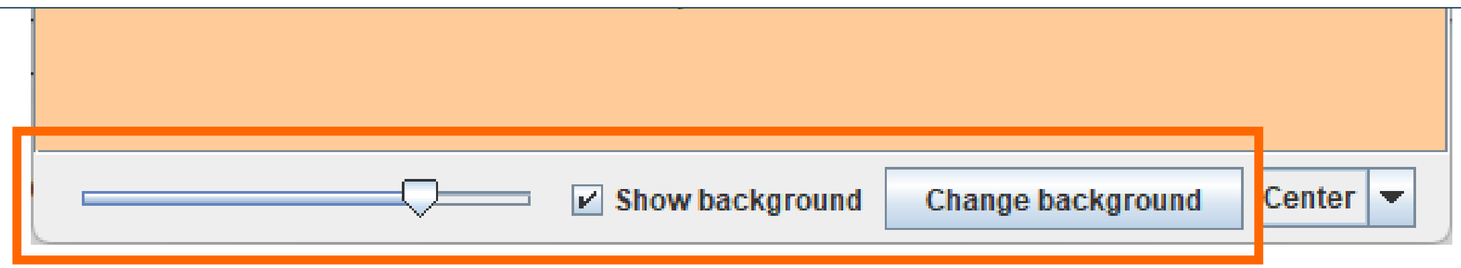
The North panel

```
JPanel northPanel = new JPanel(new GridBagLayout());
JLabel textLabel = new JLabel("Label text:");
northPanel.add(textLabel, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0, 4,0, 0), 0, 0));
JTextField textField = new JTextField(30);
textField.addActionListener(e -> label.setText(textField.getText()));
northPanel.add(textField, new GridBagConstraints(1, 0, 1, 1, 1.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(4, 4, 4, 4), 0, 0));
```



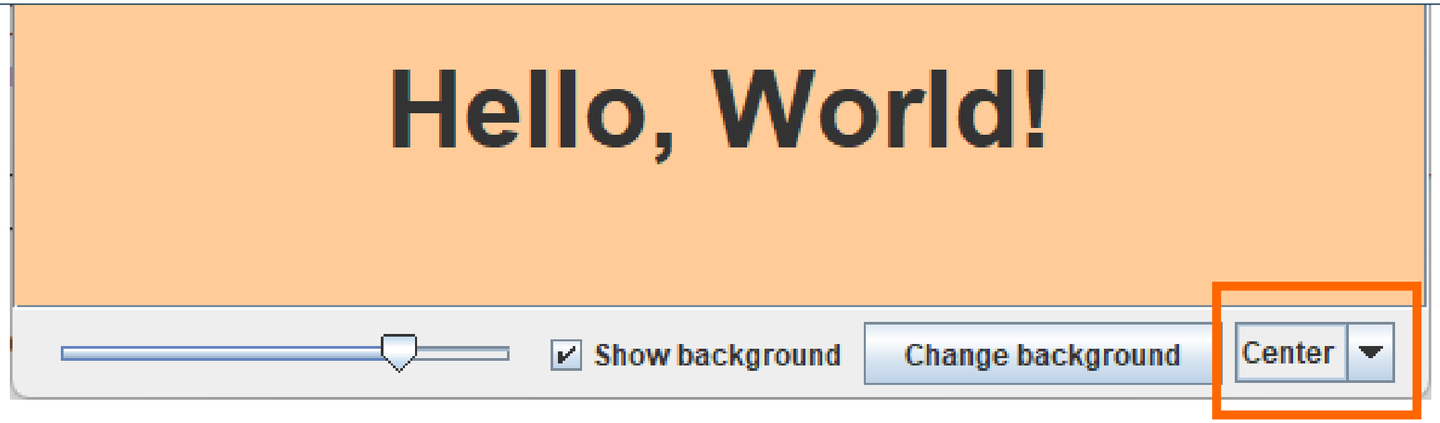
The South panel 1/2

```
JPanel southPanel = new JPanel(new FlowLayout());
JSlider sizeSlider = new JSlider(SwingConstants.HORIZONTAL, 1, 60, label.getFont().getSize());
sizeSlider.addChangeListener(e -> label.setFont(label.getFont().deriveFont((float) sizeSlider.getValue())));
southPanel.add(sizeSlider);

JButton changeColorButton = new JButton("Change background");
JCheckBox showBackground = new JCheckBox("Show background");

showBackground.addActionListener(e -> {
  label.setOpaque(showBackground.isSelected());
  label.repaint();
  changeColorButton.setEnabled(showBackground.isSelected());
});
southPanel.add(showBackground);

changeColorButton.setEnabled(false);
changeColorButton.addActionListener(e -> {
  label.setBackground(JColorChooser.showDialog(frame, "Choose background color", label.getBackground()));
});
southPanel.add(changeColorButton);
```



The South panel 2/2

```
JComboBox<Integer> alignmentComboBox = new JComboBox<>(
        new Integer[]{SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT});

alignmentComboBox.setRenderer(new DefaultListCellRenderer() {
  @Override
  public Component getListCellRendererComponent(JList<?> list, Object value, int index, boolean isSelected, boolean cellHasFocus) {
    switch ((Integer) value) {
      case SwingConstants.LEFT -> value = "Left";
      case SwingConstants.CENTER -> value = "Center";
      case SwingConstants.RIGHT -> value = "Right";
    }
    return super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);
  }
});
alignmentComboBox.setSelectedItem(label.getHorizontalAlignment());
alignmentComboBox.addActionListener(e -> {
  label.setHorizontalAlignment((Integer) alignmentComboBox.getSelectedItem());
});

southPanel.add(alignmentComboBox);
```



Look-and-feel

Swing allows to change the look-and-feel (L&F) of GUI applications, to adapt the appearance and the behavior of GUI components

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

or

```
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
```

https://www.oracle.com/java/technologies/a-swing-architecture.html

Take aways

- ❑ *Swing is not thread safe*
- ❑ *Swing documentation indicates what methods are thread-safe*
- ❑ *Thread-unsafe methods must be invoked from the event dispatch thread*
- ❑ *Components fire events in response to user actions*
- ❑ *Swing has a rich and comprehensive set of components*
- ❑ *Swing supports multiple look-and-feels*

Writing custom Swing components

Writing custom Swing components

When writing a custom swing component, you must consider the following responsibilities

- *Painting*
- *Response to GUI events*
- *Size preferences (preferred/minimum/maximum)*

PaintDemo.java

```
public class PaintDemo extends JComponent {

    public PaintDemo() {
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D scratch = (Graphics2D) g.create();
        try {
            Dimension size = getSize();
            scratch.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            scratch.drawLine(0, 0, size.width, size.height);
            scratch.drawLine(0, size.height, size.width, 0);
        } finally {
            scratch.dispose();
        }
    }

    static void main() {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                JFrame frame = new JFrame();
                frame.getContentPane().add(new PaintDemo());
                frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
                frame.setSize(200, 200);
                frame.setVisible(true);
            }
        });
    }
}
```

Painting

1. *Extend JComponent*
2. *Override paintComponent()*
    1. *create a new Graphics and remember to dispose it*

https://www.oracle.com/java/technologies/painting.html

Graphics2D

- *Control the coordinate system, through affine transformation*
- *Rendering*
  - *Shapes*
  - *Text*
  - *Images*
- *Control rendering attributes*
  - *Paint*
  - *Font*
  - *Stroke*
  - *Composite*
  - *Clip*

Repainting

When does paint happen?

- *It happens in the Event Dispatch Thread*
- *You cannot decide when*
- *You can inform Swing that a component (or a part of that component) should be repainted*
  - `public void repaint()`
  - `public void repaint(int x, int y, int width, int height)`

Event management

Swing events

- *Component events*
- *Focus events*
- *Hierarchy events*
- *Input method events*
- *Key events*
- *Mouse events*
- *Mouse motion events*
- *Mouse wheel events*
- *Window events*
- *… look at subclasses of AWTEvent*

To manage events

- *Register listeners*
- *Or enable event and override processXXXEvent*

EventDemo.java

```
public class EventDemo extends JComponent {

    private boolean pressed;

    public EventDemo() {
        enableEvents(MOUSE_EVENT_MASK | MOUSE_MOTION_EVENT_MASK);
    }

    @Override
    protected void processMouseEvent(MouseEvent e) {
        switch (e.getID()) {
            case MouseEvent.MOUSE_PRESSED -> {
                pressed = true;
                repaint();
            }
            case MouseEvent.MOUSE_RELEASED -> {
                pressed = false;
                repaint();
            }
        }
        super.processMouseEvent(e);
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D scratch = (Graphics2D) g.create();
        try {
            scratch.setPaint(pressed ? Color.YELLOW : Color.BLUE);
            scratch.fillRect(0, 0, getWidth(), getHeight());
        } finally {
            g.dispose();
        }
    }
}
```

AnimationDemo.java 1/2

```
public class AnimationDemo extends JComponent {

    private Dimension speed;
    private Point2D position;
    private Timer timer;

    public AnimationDemo() {
        position = new Point2D.Double(0, 0);
        speed = new Dimension(1, 1);
        timer = new Timer(50, this::update);
    }

    public void start() {
        timer.start();
    }

    public void stop() {
        timer.stop();
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D scratch = (Graphics2D) g.create();
        try {
            scratch.fillOval((int) position.getX(), (int) position.getY(), 3, 3);
        } finally {
            scratch.dispose();
        }
    }

    …

}
```

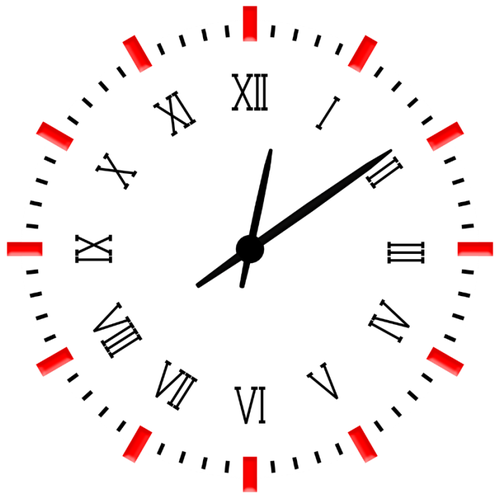
AnimationDemo.java 2/2

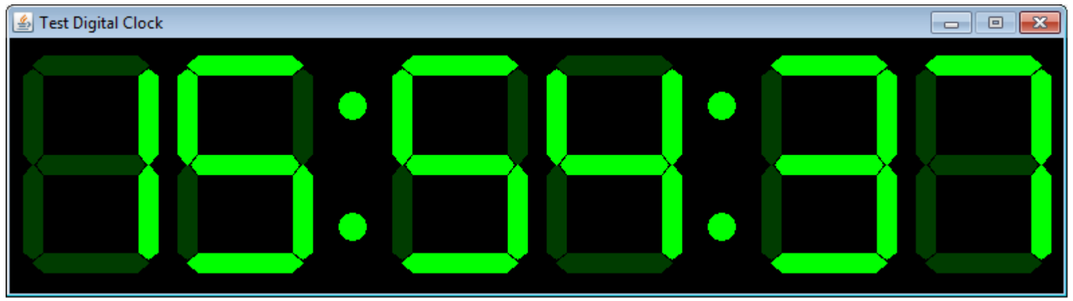
```
public class AnimationDemo extends JComponent {

    public void update(ActionEvent e) {
        position.setLocation(position.getX() + speed.width, position.getY() + speed.height);
        if (position.getX() < 0) {
            speed.width = -speed.width;
            position.setLocation(-position.getX(), position.getY());
        } else if (position.getX() > getWidth()) {
            speed.width = -speed.width;
            position.setLocation(getWidth() - position.getX(), position.getY());
        }
        if (position.getY() < 0) {
            speed.height = -speed.height;
            position.setLocation(position.getX(), -position.getY());
        } else if (position.getY() > getHeight()) {
            speed.height = -speed.height;
            position.setLocation(position.getX(), getHeight() - position.getY());
        }
        repaint();
    }

    static void main() {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Animation demo");
            AnimationDemo animationDemo = new AnimationDemo();
            frame.getContentPane().add(animationDemo);
            frame.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
            frame.addWindowListener(new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent e) {
                    animationDemo.stop();
                    frame.dispose();
                }});
            frame.setSize(200, 200);
            frame.setVisible(true);
        });
    }
}
```

Exercises

Exercise 1

Implement an analog and/or a digital clock





Self assessment

Quiz 1 – Basics of multithreading

1. **Which element is *not* part of a thread's state?**
   A. Stack of calls
   B. Program counter
   C. Thread ID
   D. Dedicated CPU

2. **In Java, (platform) threads are scheduled by:**
   A. The JVM
   B. The OS scheduler
   C. The JIT compiler
   D. The Java thread pool

3. **The `run()` method for a `Runnable`:**
   A. Must return a value
   B. Must throw exceptions
   C. Contains the code executed by the thread
   D. Automatically starts the thread

Quiz 1 – Basics of multithreading

4. **Which method actually *starts* a thread?**
   A. `run()`
   B. `execute()`
   C. `begin()`
   D. `start()`

5. **If `thread.join()` is called:**
   A. The calling thread waits for that thread to terminate
   B. The target thread waits
   C. Both threads terminate
   D. A deadlock occurs

6. **Writing concurrent programs is hard mainly because:**
   A. Threads are slow
   B. The JVM limits thread creation
   C. Shared memory leads to race conditions
   D. Java threads lack APIs

Quiz 1 – Basics of multithreading

7. **Which technology was introduced in Java 5?**
   A. Virtual threads
   B. Fork/Join
   C. Executors and synchronizers
   D. Platform threads

8. **Parallel streams were introduced in:**
   A. Java 8
   B. Java 5
   C. Java 9
   D. Java 11

9. **3. Virtual threads were introduced in:**
   A. Java 8
   B. Java 14
   C. Java 17
   D. Java 21

Quiz 1 -Basics of multithreading

10. Platform threads are:

A. Implemented entirely by JVM
B. Lightweight threads
C. Thin wrappers around OS threads
D. Deprecated

11. Why are virtual threads useful for high-throughput apps?

A. They use GPUs
B. They avoid tying up scarce OS threads during blocking I/O
C. They always run faster
D. They use more memory

12. What manages the stop/resume of virtual threads?

A. OS
B. JVM
C. Hardware
D. Compilers

Quiz 2 – Working with Swing components

1. **Swing components should be updated from**
   A. Any thread
   B. Worker threads only
   C. Event dispatch thread
   D. OS thread
2. **Swing is not thread-safe because:**
   A. Swing components do not serialize access to their internal critical sections
   B. Swing components rely on immutable state
   C. The JVM prevents synchronization in GUI code
   D. Event handling is delegated to the operating system
3. **Long tasks in Swing should be executed using:**
   A. Executors
   B. SwingWorker
   C. Platform thread
   D. Virtual thread

Quiz 2 – Working with Swing components

4. **To check if you are on the EDT, call:**
   A. `Thread.isEDT()`
   B. `SwingUtilities.checkEDT()`
   C. `SwingUtilities.isEventDispatchThread()`
   D. `SwingThread.isActive()`

5. **What happens if EDT tasks take too long?**
   A. JVM crashes
   B. GUI becomes unresponsive
   C. OS kills the window
   D. Thread priority is increased

6. `JButton` **converts low-level mouse events into:**
   A. Action events
   B. Key events
   C. Window events
   D. Focus events

Quiz 2 – Working with Swing components

7. **Events are processed by:**
   A. Main thread
   B. EDT
   C. Arbitrary worker thread
   D. OS kernel thread

8. **To receive component events, you must:**
   A. Implement `ThreadListener`
   B. Register the appropriate listener
   C. Override `run()`
   D. Modify the event queue

9. **4. GUI events from the OS are first dispatched to:**
   A. `JComponents`
   B. Event queue
   C. Event filters
   D. All listeners directly

Quiz 2 – Working with Swing components

10. `AWTEvent` **subclasses correspond to:**
    A. Input and window events types
    B. Graphical themes
    C. Swing look-and-feel types
    D. JVM memory events

11. **The main container for Swing windows is:**
    A. `JFrame`
    B. `JPanel`
    C. `Container`
    D. `WindowManager`

12. `JScrollPane` **adds scrollbars when:**
    A. The parent container is set to resizable
    B. A button is pressed
    C. The user requests it
    D. The component does not fit its viewport

Quiz 2 – Working with Swing components

13. Which layout manager arranges components in five areas?

A. `FlowLayout`
B. `GridLayout`
C. `BorderLayout`
D. `GridBagLayout`

14. Look-and-feel affects:

A. Behavior only
B. Appearance only
C. Appearance and behavior
D. Only event processing

Quiz 3 – Writing custom swing components

1. **To create a custom Swing component, extend:**
   A. `JFrame`
   B. `JPanel`
   C. `JComponent`
   D. `SwingComponent`

2. **Custom drawing is performed in:**
   A. `paintComponent()`
   B. `paint()`
   C. `drawComponent()`
   D. `onRender()`

3. **Which class provides advanced rendering features?**
   A. `Graphics`
   B. `Graphics2D`
   C. `Graphics3D`
   D. `SwingGraphics`

Quiz 3 – Writing custom swing components

4. **To listen for mouse events directly in the component, you may:**
   A. Edit the event queue manually
   B. Modify the EDT
   C. Override `run()`
   D. Override `processMouseEvent`

Correct answers

Quiz 1

1. *D*
2. *B*
3. *C*
4. *D*
5. *B*
6. *C*
7. *C*
8. *A*
9. *D*

10.A
11.B
12.B

Quiz 2

1. *C*
2. *A*
3. *B*
4. *C*
5. *B*
6. *A*
7. *B*
8. *B*
9. *B*

10.A
11.A
12.D
13.C
14.C

Quiz 3

1. *C*
2. *A*
3. *B*
4. *D*

Thank you!

esteco.com