Have you started PSA6?
A. I haven’t looked at it at all
B. I’ve started: I’ve read over the assignment and started thinking about it
C. I’ve started writing some code
D. I’m done (moving on to extra credit)

Listeners and Events in a very simple GUI

EXAM 3 Review Session:
Thursday: 3:30-4:30, CSE 4140
If you have class at that time you can come by my office at 3 and we can do a little review then.

Great show, but I was up waaay past my bedtime. 😊
Exam 3: Next Monday (!!)

• Review in this Friday’s discussion section (and some on Wednesday as well)
• Topics/skills:
  – Write a class that extends another class
  – Write a method that overrides a method in a base class
  – Reason about what will happen when calling methods in subclass/super-class objects
  – Describe how Java uses the type of the object at run-time to decide which method to call (Polymorphism)
  – Know when a reference to an object is legal, and when it will cause a compile error (e.g. a Person reference pointing to a Student object vs. a Student reference pointing to a Person object)
  – Identify and describe situations where casting is needed (and safe, or not safe)
  – Use the Graphics object to draw simple scenes
  – Describe how the paintComponent method is used to provide custom graphics in GUIs
1. Give some examples of a user interface “event”.

A. Calling a constructor, instantiating a class.

B. Displaying images, text, and buttons.

C. Loops, if statements, returns.

D. Mouse clicks, keystrokes, mouse movements.
2. Which of the following interfaces does an object need to implement to be a **listener** for an event on a button?

A. JButton

B. ActionListener

C. JPanel

D. ActionEvent

E. JComponent
3. You have instantiated a JButton called jbt, and an ActionListener called listener. Write a line of Java code to register the listener with the JButton.

A. listener.addActionListener(jbt);

B. jbt.addActionListener(listener);

C. jbt = new JButton(listener);

D. listener = new ActionListener(jbt);
PSA6: Connect 4 GUI

Two new central concepts:
1. **Today**: Writing and registering Listener objects (Event-driven programming)
2. **Friday**: Inner classes
   (Not to be confused with subclasses!!)

Today and Friday we will look at code that is *similar* to the code you will work with and write for your PSA. However, today’s code will be different in a significant way: it does not use inner classes. You can still get started on PSA6, but make sure you do the reading for Friday first.
public ConnectFourSimpleDemo()
{
    this.turn = 'X';
    JButton jbtReset = new JButton( "New Game" );
    JButton jbtSwitch = new JButton( "Switch turns" );

    JPanel buttonPanel = new JPanel();
    buttonPanel.add( jbtReset );
    buttonPanel.add( jbtSwitch );

    this.status = new JLabel( "Welcome to Connect 4!  Turn is: " + this.turn );

    JPanel displayBoard = new JPanel();
    JPanel p1 = new MyPanel();
    JPanel p2 = new MyPanel();
    JPanel p3 = new MyPanel();

    displayBoard.setLayout( new GridLayout( 1, 3 ) );
    displayBoard.add( p1 );
    displayBoard.add( p2 );
    displayBoard.add( p3 );

    this.setLayout( new BorderLayout() );
    add( this.status, BorderLayout.NORTH );
    add( displayBoard, BorderLayout.CENTER );
    add( buttonPanel, BorderLayout.SOUTH );

    pack();
    setVisible( true );
}

Sketch what this GUI will look like
public ConnectFourSimpleDemo()
{
    this.turn = 'X';
    JButton jbtReset = new JButton( "New Game" );
    JButton jbtSwitch = new JButton( "Switch turns" );

    JPanel buttonPanel = new JPanel();
    buttonPanel.add( jbtReset );
    buttonPanel.add( jbtSwitch );

    this.status = new JLabel( "Welcome to Connect 4!  Turn is: " + this.turn );
    JPanel displayBoard = new JPanel();
    JPanel p1 = new MyPanel();
    JPanel p2 = new MyPanel();
    JPanel p3 = new MyPanel();
    displayBoard.setLayout( new GridLayout( 1, 3 ) );
    displayBoard.add( p1 );
    displayBoard.add( p2 );
    displayBoard.add( p3 );

    this.setLayout( new BorderLayout() );
    add( this.status, BorderLayout.NORTH );
    add( displayBoard, BorderLayout.CENTER );
    add( buttonPanel, BorderLayout.SOUTH );

    pack();
    setVisible( true );
}

class MyPanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        g.setColor( Color.yellow );
        g.fillOval( 0, 0, getWidth(), getHeight() );
    }

    public Dimension getPreferredSize()
    {
        return new Dimension( 100, 200 );
    }
}
What will happen when the user clicks the “Switch turns” button?

A. Nothing
B. An error
C. Java will print a message
D. The turn variable will switch from ‘X’ to ‘O’
E. Something else
Writing a Listener

class ButtonListener
{
    public void actionPerformed( ActionEvent e )
    {
        System.out.println( "Button clicked!" );
    }
}

What should go in the blank to complete the class that will listen for button click events?
A. extends ActionListener
B. implements ActionListener
C. extends Listener
D. implements Listener
E. You don’t need to put anything in the blank
class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
}

When a button is clicked, Java generates an ActionEvent on the button. It automatically calls the actionPerformed method on the listener(s) registered with that button.

“Hey! I’ve got an ActionEvent! Who’s going to handle it?”
class ButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Button clicked!");
    }
}

By implementing the ActionListener interface, our ButtonListener object is a ActionListener. Therefore Java knows that a ButtonListener will have implemented the actionPerformed method and it can call that method automatically.

“OK, I found an object of type ActionListener. I’ll call its actionPerformed method”
public ConnectFourSimpleDemo()
{
    this.turn = 'X';
    JButton jbtReset = new JButton( "New Game" );
    JButton jbtSwitch = new JButton( "Switch turns" );
    ButtonListener resetListener = new ButtonListener();

    JPanel buttonPanel = new JPanel();
    buttonPanel.add( jbtReset );
    buttonPanel.add( jbtSwitch );

    ...
Adding a Listener

```java
public ConnectFourSimpleDemo()
{
    this.turn = 'X';
    JButton jbtReset = new JButton( "New Game" );
    JButton jbtSwitch = new JButton( "Switch turns" );
    ButtonListener bl = new ButtonListener();
    jbtReset.addActionListener( bl );
    jbtSwitch.addActionListener( bl );

    JPanel buttonPanel = new JPanel();
    buttonPanel.add( jbtReset );
    buttonPanel.add( jbtSwitch );

    ...
```
public ConnectFourSimpleDemo()
{
    ...
    JPanel displayBoard = new JPanel();
    JPanel p1 = new MyPanel();
    pl.addMouseListene( new PanelClickListe( ) );
    ...
}

class PanelClickListe implements MouseListe
{
    public void mouseClicked( MouseEvent e )
    {
        System.out.println( "Clicked on a label" );
    }
}

What will our GUI do if we add the code above?
A. Nothing it didn’t already do
B. It will print “Clicked on a label” when we click on the leftmost yellow oval.
C. It will print “Clicked on a label” when we click on any of the yellow ovals.
D. Nothing, there is an error with this code.
implements is a promise to implement everything in the interface

class PanelClickListener implements MouseListener {
    public void mouseClicked(MouseEvent e) {
        System.out.println( "Clicked on a label" );
    }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}

When you register the MouseListener (i.e. PanelClickListener) with the panel, Java thinks it can call ANY and ALL of these methods. And it will, so you need to tell Java what to do when it calls them (even if the answer is “do nothing”).
Making our GUI more interesting

Each oval represents a "space" in a 3-position "board". Yellow indicates that no one has played.

If it is 'X's turn, clicks should turn the ovals blue.

If it is 'O's turn, clicks should turn the ovals green.

Turns will switch only when the user clicks "Switch turns".

Idea: We will create a new instance variable "board" and the Panels will paint based on the contents of the "board" variable.
public class ConnectFourSimpleDemo extends JFrame {
    private char turn;
    private char[] board;
    private JLabel status;

    public ConnectFourSimpleDemo() {
        this.turn = 'X';  // Initialize turn to X

        this.board = new char[3];  // Initialize the board (empty)
        this.board[0] = ' ';  
        this.board[1] = ' ';  
        this.board[2] = ' ';  

        JPanel displayBoard = makeDisplayBoard();
        JPanel buttonPanel = makeButtonPanel();

        this.status = new JLabel( "Welcome to Connect 4!  Turn is: " + this.turn );
        setLayout( new BorderLayout() );
        add( this.status, BorderLayout.NORTH );
        add( displayBoard, BorderLayout.CENTER );
        add( buttonPanel, BorderLayout.SOUTH );

        pack();
        setVisible( true );
    }
}

Our method was too long, so I added helper methods
public class ConnectFourSimpleDemo extends JFrame {
    // ConnectFourSimpleDemo is a completely separate class, even though
    // both classes are defined in the same file.
}

class MyPanel extends JPanel {
    private int position;

    public MyPanel( int pos )
    {
        position = pos;
    }

    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        if ( board[position] == 'X' )
            g.setColor( Color.blue );
        else if ( board[position] == 'O' )
            g.setColor( Color.green );
        else
            g.setColor( Color.yellow );
        g.fillOval( 0, 0, getWidth(), getHeight() );
    }
}

Will this work?
A. Yes
B. No, because you cannot access the ConnectFourSimpleDemo member variable board from the MyPanel class
C. No, because the MyPanel constructor does not explicitly call the constructor of the superclass (JPanel)
ConnectFourSimpleDemo object (scope)

- **turn**: ‘X’
- **status**: Address of the JLabel object
- **board**: 

MyPanel object (scope)

- **position**: 

No connection between MyPanel object and CFSD object!
Now if Java can’t find a variable in the MyPanel object, it will look for it in the CFSD object!

Do not confuse this with subclasses! The MyPanel object exists within the scope of the CFSD object, but there is no subclass relation. The board variable is still in the CFSD object, NOT the MyPanel object.
Now if Java can’t find a variable in the MyPanel object, it will look for it in the CFSD object!

Do not confuse this with subclasses! The MyPanel object exists within the scope of the CFSD object, but there is no subclass relation. The board variable is still in the CFSD object, NOT the MyPanel object. A MyPanel exists within a CFSD, but it is NOT a CFSD.