CSE 8B Today

Listeners and Events in a very simple GUI

PSA 6!
- Released tonight
- TODAY and THURSDAY: HELP to work on it
- Due next Wednesday!!
SEE WEB for next PSAs planning,

Value of PSA EXTRA will push your PSA grade
More than a star point for sure!
but I’m still deciding the best way.

In-term exam 3 NEXT TUESDAY

Seating requests: (NO NEED TO SEND IT AGAIN if YOU ALREADY DID)
IF YOU NEED a NEW REQUEST SUBMIT BY THURSDAY night!! WE ARE CHANGING NEXT WEEK
1. Which of the following sentences does describe one key difference between non-abstract and abstract classes.

A. We can use the “new” operator to create instances of abstract classes but we can’t do that for instances of non-abstract classes.

B. The abstract methods defined in an abstract class are not implemented in that class but they have to be implemented in any super class that is non-abstract.

C. All the methods in non-abstract classes are defined with implementation, whereas abstract classes can have methods defined without implementation.

D. Non-abstract classes can define methods without implementation, whereas abstract classes must define all of their methods with implementation.
2. The following object is instantiated (like in the book):

   GeometricObject geoObject = new Rectangle(10, 20);

   getPerimeter() is an abstract method in the GeometricObject class.
   When you call geoObject.getPerimeter(), in which class can we actually find the code that is run by the getPerimeter() method?

   A. GeometricObject
   B. Circle
   C. Rectangle
   D. Object
3. What java keyword do you use to implement an interface?

A. extends
B. final
C. implements
D. import
E. use
4. What is the name of the method specified in the Comparable interface?

A. smallerThan
B. greaterThan
C. compareTo
D. equalTo
E. there is no method specified in that interface
Exam 3: Next Tuesday(!!)

- Review at Tomorrow’s Discussion. This test will be more questions and shorter
- Focus 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). Examples where runtime errors happen?
  - Use the Graphics object to draw simple scenes
  - Describe how the paintComponent method is used to provide custom graphics in GUIs
  - Use Interfaces and Abstract classes, and describe what these are and when you would use them
  - Reason about errors when using interfaces and abstract classes

- NOT included:
  - Events
  - Inner classes
PSA6: Connect 4 GUI

PSA 6: a graphical INTERFACE FOR A GAME!
- your ConnectFourBoard class controls game play and game state (AND IT’s ALREADY DONE!)
- Your interface simply calls methods (NO NEED TO RE-IMPLEMENT the connectFourBoard class! You CAN just USE the methods we give) on the board to make moves and reflect changes in the game.

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

Today and Thursday: working on the code you will work with and write for your PSA. However, today we won’t talk much about inner classes. You can still get started on PSA6, but make sure you do the reading for Thursday first.

DEMO in Dr. Java
public class ConnectFourSimpleDemo extends JFrame {

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

    pack();
    setVisible( true );
  }

  You should be able to 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 );
    }
}
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 );
}
Writing a Listener

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?"
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.

1st step: CREATE CODE TO BE RUN WHEN THE ACTION HAPPENS!

“OK, I found an object of type `ActionListener`. I’ll call its `ActionPerformed` method”
Adding a Listener

```java
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 );

    ...  
```

If we add the line in red to the code, what will the GUI do when we click on the buttons?

A. It will print “Button clicked!” when either the New Game or the Switch turns button is clicked  
B. It will print “Button clicked!” when the New Game button is clicked, but not when the Switch turn button is clicked.  
C. Nothing will happen when either button is clicked.  
D. It will have an error.
Adding a Listener

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

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 );
   ...
Another example: Mouse Listeners

public ConnectFourSimpleDemo()
{
    ...
    JPanel displayBoard = new JPanel();
    JPanel p1 = new MyPanel();
    p1.addMouseListener( new PanelClickListener()
    );
    ...
}

class PanelClickListener implements MouseListener
{
    public void mouseClicked( MouseEvent e )
    {
        System.out.println( "Clicked on a label" );
    }
}  // end of the MouseListener class

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. (see next slide...)
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”) – IF NOT, you’ll get a compilation error! (as in previous slide)
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.

WE WANT SOME “CONNECTION” between what happens in some Components and what is shown in others
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 buttonPanel = makeButtonPanel();
        JPanel displayBoard = makeDisplayBoard();
        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.
} END of ConnectFourSimpleDemo

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)
I could pass a LOT of parameters to the constructor... instead of just public MyPanel( int pos ) BUT . . . THERE IS A BETTER SOLUTION
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 in the MyPanel object.
ConnectFourSimpleDemo object

<table>
<thead>
<tr>
<th>turn</th>
<th>‘X’</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Address of the JLabel object</td>
</tr>
<tr>
<td>board</td>
<td></td>
</tr>
</tbody>
</table>

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 in the MyPanel object.

A MyPanel exists within a CFSD, but it is NOT a CFSD.
public class ConnectFourInnerDemo extends JFrame {
    // ConnectFourInnerDemo class defined here . . .
    class MyPanel extends JPanel //inner from CFID
    {
        private int position; // A new variable in MyPanel
        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());
        }
        // other MyPanel methods defined here
    }
    // Other methods defined here

    class PanelClickListener implements MouseListener {
        //inner from MyPanel
        public void mouseClicked(MouseEvent e)
        {
            _________________________________;
            repaint();
        }
        // Other methods defined here

    }
    thankful for your effort.
    How many “inner” levels??
}
ConnectFourInnerDemo object

**Turn**: 'X'

**Status**: Address of the JLabel object

**Board**

MyPanel object

**Position**: 0

PanelClickListener object

```java
// method from MyPanel object
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());
}
```

// reads data from board

// writes data to board
public class ConnectFourInnerDemo extends JFrame {
    // CFID defined here
    class MyPanel extends JPanel
    {
        private int position;  // A new variable in MyPanel
        // MyPanel methods defined here
    }

    class PanelClickListener implements MouseListener
    {
        public void mouseClicked( MouseEvent e )
        {
            ________________________________;
            repaint();
        }
    }  // We will register this listener
    // with the MyPanel objects

    What should go in the blank to make the panel turn blue when the user clicks on it and it’s X’s turn?
    A. board[position] = turn;
    B. g.setColor( Color.blue );
    C. if ( turn == ‘X’ ) g.setColor( Color.blue );
    D. Something else

    HINT: calls to paintComponent