CSE 8B Today

Upcoming schedule:
Class:
• Monday: More recursion with Prof. Beth Simon
• Wednesday: NO CLASS (video lecture... I hope!)
PSAs:
• PSA7 due next Saturday (5/25)
• PSA8 due Thursday 6/6
• One more extra credit PSA coming! (due Friday 6/7)
Exams:
• Interm exam 4: Wednesday 5/29

Applets and Graphic letters!
1. A Java GUI program can be converted into an applet by replacing JFrame with JApplet, and deleting which method(s)?

A. main

B. The constructor

C. Getters and setters

D. All private methods
2. What language is used to run a Java applet from the browser?

A. PHP

B. HTML

C. C++

D. Python
3. JFrame and JApplet are both subclasses of which class?

A. JComponent

B. ActionListener

C. Container

D. JLabel
public class AppletDemo extends JApplet {

/** A message that will reflect what the user types */
private JLabel message;
private JTextField textfield;

public void init() {
  System.out.println( "Initializing the applet" );
  message = new JLabel( "Type something in the field below" );
  message.setFont( new Font( "Serif", Font.PLAIN, 40 ) );
  textfield = new JTextField();

  textfield.addKeyListener( new MessageListener() );

  setLayout( new BorderLayout() );
  this.setSize(new Dimension(700,500));
  add( message, BorderLayout.CENTER );
  add( textfield, BorderLayout.SOUTH );
  setVisible( true );
}

Sketch what this applet will look like.
Type something in the field below

Run with:
> appletviewer index2.html
DO NOT DEBUG IN THE BROWSER!!
**KeyEvents**

```
java.awt.event.InputEvent
```

```
java.awt.event.KeyEvent

+getKeyChar(): char
+getKeyCode(): int
```

Returns the character associated with the key in this event.
Returns the integer key code associated with the key in this event.

**Figure 16.14** The `KeyEvent` class encapsulates information about key events.
Complete the code

```java
class MessageListener implements KeyListener
{
    public void keyPressed(KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyTyped(KeyEvent e)
    {
        ________________________________
    }
}

class MessageListener implements KeyListener
{
    public void keyPressed(KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyTyped(KeyEvent e)
    {
        ________________________________
    }
}

Which line of code goes in the blank to complete the Applet? (MessageListener is an inner class of AppletDemo)
A. message.setText( "You typed a " + e.getKeyChar() );
B. System.out.println("You typed a " + e.getKeyChar() );
C. textField.keyTyped();
D. textField.setText("You typed a " + e.getKeyChar() );
```
Other than the fact that this is an Applet (no big deal) and we’re using KeyListeners, you’ve done all this before! If you get confused, stop and go seek help.
Thinking recursively

factorial

\[ N! = N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1 \]

\[ N! = N \times (N-1)! \]  Recursive step, do a tiny bit of work, and then assume you have code that solves a smaller version of the same problem.

(and 1! = 1)

Base case (what to do when the problem can’t be simplified any further)
Complete the factorial method

```java
public int fac( int n ) {
    if ( _____________(1) _____________ )
        __________(2)______________
    else
        return n * fac( n-1 );
}
```

What should go in blank (1)?
A. $n > 1$
B. $n <= 1$
C. $\text{fac}(n) == 1$
D. $\text{fac}(n) > 1$
Complete the factorial method

public int fac( int n ) {
    if ( n <= 1 )
        return 1;  // (2)
    else
        return n * fac( n-1 );
}
Recursion practice

Write a method to change all the Strings in a String array to lower case, recursively

[“Hello”, “CSE8B”] \(\rightarrow\) [“hello”, “cse8b”]

```java
public void allToLower( String[] myList, int index )
{
    ???
}
```

What is the base case? (i.e., the simplest version of the problem)
A. When the myList already contains only lower case strings
B. When index is (greater than or) equal to the myList.length
C. When index is (greater than or) equal to the myList.length-1
D. When myList is empty

B is more elegant
Recursion practice

Write a method to change all the strings in a String array to lower case, recursively
[“Hello”, “CSE8B”] → [“hello”, “cse8b”]

```java
public void allToLower(String[] myList, int index)
{
    if (index >= myList.length)
        return;
    else {
        ???
    }
}
```

What line or lines complete the method correctly?

A. `myList[index] = myList[index].toLowerCase();
    allToLower(myList, index);`  **args don't change!!**

B. `allToLower(myList, index - 1);`  **wrong way**

C. `myList[index] = myList[index].toLowerCase();
    allToLower(myList, index + 1);`  **progress towards base case**

D. `myList[index] = myList[index].toLowerCase();
    allToLower('myList, index-1);`  **wrong way**
NOTE: All recursive code can be implemented with loops

- Simple: if, else recursion is usually “easy” to turn into a loop
- Next we’ll look at a slightly more complex case (which is still pretty easy to do with a loop)
- Monday: we’ll look at some more complex recursion techniques that are harder to write with loops
  - It’s “natural” to do them recursively
Recursion: slightly harder practice

Write a method to find an element in an array and return its index

```java
public int find( String[] myList, String toFind, int currIndex )
{
    ???
}
```

What is the base case?
A. The element at currIndex is equal to toFind
B. The array is empty
C. currIndex is (greater than or) equal to myList.length
D. toFind is not in myList
E. Something else
Recursion: slightly harder practice

Write a method to find an element in an array and return its index

```java
public int find( String[] myList, String toFind, int currIndex )
{
    if ( currIndex >= myList.length )
        return ???
    if ( myList[currIndex].equals( toFind ) )
        return currIndex;

    // recursive step here
}
```
Worksheet: Complete the recursive step

Write a method to find an element in an array and return its index

```java
public int find( String[] myList, String toFind, int currIndex )
{
    if ( currIndex >= myList.length )
        return -1 // It wasn’t there!
    if ( myList[currIndex].equals( toFind ) )
        return currIndex;  // That was easy!
}
```