CIDSE
CSE 205 Review

Fulton Schools of Engineering Tutoring Centers
October 19th, 2015
Fall 2015
### Table of Contents

1) Linked list
   - Traversal
   - Insertion
   - Deletion

2) Search
   - Binary Tree & Search
   - Sorting:
     - Insertion
     - Selection
     - Merge
     - Quick

3) GUI
   - Event handling

4) Input & Output
   - I/O Exception Handling
A linked list is like a train where each car (node) is connected to the next, but are not identified by numbers.

The Linked List class can be imported from java.util. It uses the Collection interface. That means you can create an object of the ListIterator class in your linked list to create an iterator object that keeps track of your position when traversing a linked list.

The placement of the iterator when it is first created is before the first element. That means it’s .next will be the first element.

Make the data/object type of your iterator the same as the type of the list you’re traversing, ie:

```java
ListIterator<String> iterator = stringList.listIterator();
```

The above creates an iterator object for a linked list called stringList that stores strings.
Linked List: Traversing, Adding, and Removing using ListIterator

- When you use a ListIterator object you have access to its methods.
- `.next()`; `.hasNext()`;
- You can use the ListIterator to add or remove an element:
  - `.add("something")`; `.remove("something")`;
- Keep in mind where the ListIterator is currently positioned in relation to `.next()`; and `.previous()`;. It will add the new element after its position.
- If it is a new ListIterator that has not been moved at all, it will add it to the first slot of the linked list, because it is positioned before the current first element.
- If you use `.next()` one time and then `.add()`, your new element will become the second element in the linked list (assuming there are other elements in the list).
Linked List

- Want more info? Look to Chapter 15 of *Java For Everyone 2e.*
A **Binary Search Tree** is a special kind of binary tree where all the elements on the left branch of each root is smaller than the root, and all elements on the right branch are larger.

**$O(n \log n)$** to search a binary search tree
Summary of sorting algorithms:

- **Insertion**: $O(n^2)$
- **Selection**: $O(n^2)$
- **Merge**: More efficient than selection sort, $O(n\log n)$
- **Quick**: $O(n^2)$

For more information, refer to Chapter 14 of *Java For Everyone 2e*.
When the mouse is inside the window of a GUI program, events occur.

If you want your program to react to events, you need to use event listener objects.

Event Listeners are instances of classes, they provide instructions in their methods explaining how you want your program to react to events—like button clicks.

Without event listeners, your buttons/GUI elements will not respond to user interaction!

- Users can click all they want, without a listener, your buttons, lists, etc., are not paying attention.
Event listeners are the wiring and engineering behind your elements.

If you want your mechanical gizmo to jump ten feet in the air, you’re going to need a spring.

Similarly, if you want your program to change the text of a JLabel from “Ready to jump” to “jumping!” when a JButton is pressed, you need an event listener object.

You need both to create a listener object and connect your event listener with its event source—in this case, your JButton for your code to execute properly.
You implement the `ActionListener` interface to create a listener class that does what you want when the action happens—such as changing text of a `JLabel` in the example below.

This class will be a `nested private class` inside the class or code block that contains your elements.

In the code to the left, the `JLabel` named `jumpLabel` will change text to “Now jumping” when someone interacts with the element that has this listener added to it.
Event Handling Step 2: Create an Object From Your New EventListener Class

- Just like how you use the `Scanner` class, after you create your nested class—JumpListener in this case—you must create an object of that class before you can apply it to any elements.
- You can create this object any time before its needed. Make sure it’s accessible to the GUI elements that need it.

```
//before you can use a listener, you must create a listener object
//from your listener class
JumpListener jumpListener = new JumpListener();

//make sure to initialize your variables!
jumpLabel = new JLabel("ready to jump");
jumpButton = new JButton("Click to Jump");
```
Event Handling Step 3: Add Your Object to the Element

- GUI elements like JButton and JLabel have an add method so that an ActionListener can pay attention to them.
- If you don’t add your ActionListener to your object, they don’t pay any attention to each other.
- Make sure to add your ActionListener after your GUI elements have been initialized.

```java
//This is how the program knows to make stuff happen when the jump button is pressed!
jumpButton.addActionListener(jumpListener);
```
Event Handling Step 4: Test your code

- Now you should have everything in place to check if your elements are actually interacting with your ActionListener correctly.
- If you get the output you want, congratulations!
- If you’re not seeing anything on your screen, check the layouts of your JPanels, and if your elements are all added to your Jpanels.
GUI Event Handling

Want more info? Look to Chapter 10.2 in *Java for Everyone 2e*. 
Exception Handling

- When your program comes across a error it doesn’t know what to do with, it will stop.
- You can avoid the program stopping itself by giving it instructions for when it runs into errors. This is what input/output exception handling is about.
- You can even force the program to claim it has found a certain kind of error/exception.
- If you want to throw an input/output related exception (like FileNotFoundException), you will need to input java.io.
Use the `try` and `catch` keywords.

Kind of like loops, blocks of try code will try to execute, but if an exception detailed in a catch statement is found, the code will stop executing and move to the catch.

If you get a `FileNotFoundException` and have an `IOException` catch statement, this will execute because a `FileNotFoundException` is a descendant of `IOException`.

Catch blocks only execute if the exception identified in the catch statement happens during the try block of code.

The `Finally` block of code executes after the try and catch statements have executed, regardless.
If you indicate that your method throws an exception using the `throws` keyword, when the method runs into that kind of exception, the method will end.
I/O Exception Handling

- I/O exceptions can be caused by things outside the programmer’s control, such as a disk error. If you do not indicate how to deal with a checked exception like this, the program will not compile.
- Want more info? Look to Chapter 7.4 of Java For Everyone 2e.
Questions?
References

- Java For Everyone 2e