Contents

1 Nov. 6–8: Recursion and binary search 3
  1.1 Recursion, obligatory factorial example 3
  1.2 Call stack in recursive functions 4
  1.3 Searching 4
  1.4 Iterative search 5
  1.5 Recursive search 6
  1.6 Benefits of recursion 6
  1.7 Efficiency 7

2 Nov. 13: Selection sort and mergesort 8
  2.1 Sorting 8
  2.2 Selection sort 8
  2.3 Recursive selection sort 8
  2.4 Efficiency of selection sort 8
  2.5 Interlude: sorting a deck of cards 9
  2.6 Mergesort 9
  2.7 Efficiency of mergesort 10

3 Nov. 15 (part 1): Exceptions 11
  3.1 Old school error reporting 11
  3.2 Exceptions solve this problem 11
  3.3 What are exceptions? 12
  3.4 Exceptions in Java 12
  3.5 Java try-catch example 12
  3.6 A more complete example 13
  3.7 Checked versus unchecked exceptions 13
  3.8 Example: NumberFormatException 13
  3.9 Declaring exceptions 14
  3.10 Never write this 14
  3.11 Readings 14

4 Nov. 15 (part 2): Text files 15

5 Nov. 20: Regular expressions 16
  5.1 Regular expressions 16
  5.2 The Kleene star 16
  5.3 Escaping 16
  5.4 grep and egrep 17
  5.5 Exercise 17
1 Nov. 6–8: Recursion and binary search

1.1 Recursion, obligatory factorial example

• It is a tradition to explain recursion in terms of factorials.
• Who am I to challenge tradition?
• Usual implementation looks like this:

```c
int factorial(int x) {
    int answer = 1;
    for (int k = 1; k <= x; k++) {
        answer *= k;
    }
    return answer;
}
```

• If \( x \geq 1 \), it is the case that \( \text{factorial}(x) = x \times \text{factorial}(x - 1) \).
• We can use this to implement it:

```c
int recursiveFactorial(int x) {
    if (x == 0) { // base case
        return 1;
    } else {     // inductive/recursive case
        return x * recursiveFactorial(x - 1);
    }
}
```

• In the “base case,” the result is simple.
• In the “recursive case” or “inductive case,” the result is expressed in terms of a simpler result.
1.2 Call stack in recursive functions

Example session from RecursiveFactorial.java:

```
$ java RecursiveFactorial 9
java.lang.Exception: Stack trace
at java.lang.Thread.dumpStack(Thread.java:1273)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:9)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.recursiveFactorial(RecursiveFactorial.java:12)
at RecursiveFactorial.main(RecursiveFactorial.java:21)
9! = 362880
```

1.3 Searching

- Enough of the toy examples, here’s a practical one.

- Easiest way to find an object in a `List`:

  ```java
  // Returns true if array contains elem, false otherwise.
  public <T> boolean arrayContains(T[] array, T elem) {
    for (int i = 0; i < array.length; i++) {
      if (array[i].equals(elem)) {
        return true;
      }
    }
    return false;
  }
  ```

- If the array is unsorted, we have to do it this way.

- If the array is sorted, we can do a binary search.

- If the middle element of the array is greater than the element we’re searching for, we know the element is in the lower half.

- Similar for the upper half.

- If it’s equal, we win!
1.4 Iterative search

To keep things interesting, let’s do it the hard way first (see also arrayContains2 in ArrayContains.java):

```java
// Returns true if array contains elem, false otherwise.
// Since the array is sorted, we take a Comparator that
// tells us how the elements were sorted.
public <T> boolean arrayContains2(T[] array, T elem,
                                 Comparator<T> order) {
    // First element that could match.
    int lower = 0;
    // Length of sub-array that could contain a match.
    int length = array.length;
    while (length > 0) {
        // Find an element halfway through. / rounds down, so
        // since the length is positive, this is inside the
        // array we’re searching.
        int pivot = lower + length / 2;
        int c = order.compare(elem, array[pivot]);
        int elementsBeforePivot = pivot - lower;
        if (c < 0) {
            // Element before pivot, shrink sub-array to the left.
            length = elementsBeforePivot;
        } else if (c > 0) {
            // Element after pivot, move lower to the right and shrink.
            lower = pivot + 1;
            length -= elementsBeforePivot + 1;
        } else {
            // Found it.
            return true;
        }
    } // end while
    // Shrunk to a zero-length array, so elem was not found.
    return false;
}
```
1.5 Recursive search

Here’s the same algorithm implemented with recursion (see also arrayContains3 in ArrayContains.java):

```java
public <T> boolean arrayContains3(T[] array, T elem, Comparator<T> order) {
    return arrayContainsRecurse(array, elem, order, 0, array.length);
}

public <T> boolean arrayContainsRecurse(T[] array, T elem, Comparator<T> order, int lower, int length) {
    if (length == 0) {
        return false;
    }
    int pivot = lower + length / 2;
    int c = order.compare(elem, array[pivot]);
    if (c == 0) {
        return true;
    } else if (c < 0) {
        return arrayContainsRecurse(array, elem, order, lower, pivot - lower);
    } else {
        return arrayContainsRecurse(array, elem, order, pivot + 1, length - (pivot - lower + 1));
    }
}
```

1.6 Benefits of recursion

- Both versions (if you leave out the comments) are about the same size.
- Main benefit of recursion is in thinking about it.
- Many algorithms are easier to think about this way.
- In binary search, one subproblem ↔ one function call.
1.7 Efficiency

- More on this in CSCI-UA.0102
- For now, notice that in the sequential search, we had to look at every element.
- In the binary search, the sub-array gets cut in half each call to `arrayContainsRecurse`.
- That means that we can search through twice as many objects with only one call.
- So, it takes one call to search through zero items, 2 for 1 item, and one more each time we double the number of items we can search through, so, roughly: items \(\approx 2 \cdot \text{calls}\).
- Or, equivalently: calls \(\approx \log_2\) items.
- It’s not exact, since we are actually dividing the array into three parts, not two: the elements before the pivot, the elements after the pivot, and the pivot.
- But, it’s pretty close.
- Consider how making subproblems and function calls the same thing with recursion makes analyzing the efficiency easier.
- Log base two is pretty good.
- You can’t do better if all you know is that the elements are ordered.
2 Nov. 13: Selection sort and mergesort

2.1 Sorting
- Last time we talked about sequential search and binary search.
- In sequential search, we searched an array by looking at every element.
- In binary search, we assume the array is sorted.
- We take advantage of this to chop it in half successively until we find the part with the element we’re looking for.
- But, how do the elements get sorted in the first place?

2.2 Selection sort
- Selection sort is the sorting analog of sequential search.
- Find the largest element, put it in the last position.
- Find the largest element remaining, put it in the second-to-last position.
- Repeat until done.
- Can be implemented recursively or iteratively. Let’s do recursive.

2.3 Recursive selection sort
- Given length elements, find the largest and put it in position length - 1.
- Sort first length - 1 elements.
- Demo in SelectionSort.java.

2.4 Efficiency of selection sort
- There are approximately $array.length$ calls to selectionSortRecurse.
- Each call calls compare a bunch of times, $array.length - 1$ the first time through, only one when sorting the first two elements, and about $array.length / 2$ when it’s halfway done.
- On average, it has to look at half the elements each call to selectionSortRecurse.
- So, there are about $array.length * array.length / 2$ total calls to compare.
• We call this an $O(n^2)$ algorithm since to within a constant factor, it takes $n^2$ operations to handle a data set of size $n$.

• Similarly, sequential search is $O(n)$ and binary search is $O(\log n)$.

2.5 *Interlude: sorting a deck of cards*

• Bubble sort (Homework 4):
  – Flip through deck.
  – Every time you see a pair of cards out of order, switch them.
  – Repeat until all cards are in order.

• Selection sort:
  – Find the first card, put it aside.
  – Find the next card, put it on top of the first.
  – Repeat until all cards are in the sorted pile.

• Mergesort:
  – Split the cards in half.
  – Split the first pile in half.
  – Split the first half-pile in half again, repeating until the piles only have one card.
  – Combine the small piles by picking up the smallest card in a pile until all the cards have been combined.
  – Move on to the next pile.
  – Repeat until the two piles comprising the entire deck are merged.

2.6 *Mergesort*

• We can do better than selection sort or bubble sort!

• Let’s divide up the problem like we did with binary search.

• Sort the first half of the array, sort the second half, and merge the results together.

• You can implement mergesort iteratively, but it is hard to get right.

• n.b. “Sequential search,” “binary search,” and “selection sort” are all two word names. “Mergesort” is a single word.

• Demo in Mergesort.java.
2.7 Efficiency of mergesort

- Mergesort splits the array in half until the arrays are only two long, so just like binary search, it calls itself to a depth of log base two of the number of elements in the array.

- Unlike binary search, it can’t throw away half the array every time it calls itself.

- At every depth, each element gets copied once and merged once, so each layer of recursion requires about array.length operations.

- Multiplying the length times the log of the length gives us the total number of operations to within a constant factor.

- Or, concisely, mergesort is $O(n \log n)$.

- This is much better than selection sort, which is $O(n^2)$.

- As much better than selection sort as binary search is better than sequential search.

- If all you can do is compare elements, you can’t sort in better than $O(n \log n)$ time.

- You can avoid taking up extra space with temporary arrays with quicksort, which will be covered in CSCI-UA.0102.

- Quicksort is really neat.
3 Nov. 15 (part 1): Exceptions

3.1 Old school error reporting

• How should a method indicate to its caller that something went wrong?
• All of these patterns are used by high-quality code.
• For example, suppose a method reads a file. What if the named file isn’t there?
• It could abort the program.
  – What if the file is optional?
  – What if special error handling or logging is necessary?
• It could return a boolean indicating success or failure and return the contents of the file by modifying an argument.
  – This makes for long-winded code since methods can’t be called with the return values of other methods.
  – What if the caller forgets to check the return value?
• It could return the desired return type and return an error code by modifying an argument.
  – Similarly long-winded code.
  – Similar problems with checking the return value.

3.2 Exceptions solve this problem

• Method gets to return a regular value.
• If something goes wrong, special error handling code is invoked.
• If there is no error handling code, the caller’s caller gets a chance, and so on up the call stack.
• If no method handles the exception, the program aborts.
3.3 What are exceptions?

- Exceptions are alternate return paths for a method.
- Usually a method returns an X, but occasionally returns a Y.
  
  - “Usually a reciprocal method returns a number. Occasionally it returns a divide-by-zero error.”
  - “Usually a file-reading method returns the contents of a file. Occasionally it returns a file was not found error.”
  - “Usually a sorting method returns nothing at all and modifies its arguments. Occasionally it returns an an error indicating that the objects to be sorted are not comparable.”

- Take a separate path through the code, so that caller can’t ignore it.
- Should only be used for unexpected conditions.
  
  - Bad: reached the end of a file
  - Good: data corruption detected
  - Bad: sort function passed a zero-length array
  - Good: sort function passed a null array

3.4 Exceptions in Java

- **`throw`** is like **`return`** but returns an exception instead of a regular value.
- **`try`** and **`catch`** are used instead of assignment to handle the case where an exception is thrown.
- Just as a return type says what the type of the regular return value is, **`throw`** is used instead of a return type to indicate what kinds of exceptions a method may throw.
- **`finally`** lets you run some common code regardless of whether an exception is thrown.

3.5 Java try-catch example

```java
try {
    System.out.print("Enter a positive number to ponder: ");
    int n = (new Scanner(System.in)).nextInt();
    // Do something with n here.
} catch (InputMismatchException e) {
    System.out.println("That does not look like a number.");
}
```
3.6 A more complete example

See ExceptionDemo.java.

3.7 Checked versus unchecked exceptions

- In Java, you can throw any object that is a subclass of Throwable.
- Exception and Error are the two main subclasses.
- Error indicates a Java-wide problem, like running out of memory.
  - You rarely want to catch these.
  - Let them propagate up to the to level and crash the program.
- RuntimeException is a subclass of Exception.
  - You occasionally want to catch these, but they’re usually indicative of a bug.
  - Throw these if a caller might want to handle the exception, but probably doesn’t.
- Exception (other than RuntimeException) is the superclass of all checked exceptions.
  - Throw these when it is critical that the caller have error-handling code.
  - Cannot throw a FooException with “throw new FooException(...)” unless the method has “throws FooException” in its definition.

3.8 Example: NumberFormatException

- NumberFormatException is thrown by Integer.parseInt if you pass it a String that is not a valid integer.
- If you generated the argument yourself, it indicates a bug, so you don’t want to catch the exception.
- If the argument was user input, it may indicate bad user input, which is expected, so you want to catch it.
- NumberFormatException is a RuntimeException, so you get the choice.
3.9 Declaring exceptions

- Even though unchecked exceptions do not need to be explicitly declared, it's good form.
- Exceptions thrown are every bit as much of the function's contract as the return value.
- Can’t expect a caller to ever handle an unchecked exception if they don’t know about it.
- Example:

  ```java
  public static int parseInt(String s)
  throws NumberFormatException {
      ... 
  }
  ```

3.10 Never write this

You may be tempted to write something like this to get the Java compiler to shut up about checked exceptions that can’t ever happen:

```java
try {
    // some stuff that throws an exception
} catch (Exception e) {
}
```

Never do that. It’s impossible to debug when the impossible things happen. Do this instead:

```java
try {
    // some stuff that throws an exception
} catch (IOException e) {
    // Turn the specific exception that can’t happen into an unchecked exception, so if it does happen, your program aborts and you get an error message you can use for debugging.
    throw new RuntimeException(e);
}
```

3.11 Readings

4 Nov. 15 (part 2): Text files

- Reading and writing text files is boring. Let’s do this quickly.

- The `java.io.File` class is a reference to a file or directory on disk.

- Creating one with `new` does not create the file, nor does it get deleted when the object goes away.

- `java.util.Scanner` has a constructor that takes a `java.io.File`.
  - Use this to read plain text files.

- `java.io.PrintWriter` has a constructor that takes a `java.io.File`.
  - Use this to write plain text files.
  - `PrintWriter` is very similar to `java.io.PrintStream`.
  - `System.out`, which we have been using the whole course, is a `PrintStream`.

- Example in `FileDemo.java`.
5 Nov. 20: Regular expressions

5.1 Regular expressions

- On to the good stuff!
- This section is not about Java.
- Regular expressions are text strings that match patterns in text.
- Most characters match themselves.
  - Ex.: “foo” matches “foo”.
- A single period matches any character.
  - Ex.: “ba.” matches both “bar” and “baz”.
- A vertical bar separates alternatives.
  - Ex.: “foo|bar” matches both “foo” and “bar”.
- Parentheses group alternatives.
  - Ex.: “f(r|o)oba(r|z)” matches both “foobar”, “frobar”, “foobaz”, and “frobaz”.

5.2 The Kleene star

- Adding a * (asterisk) to a character or group says “zero or more of that group.”
- Called the “Kleene star” after its inventor, Stephen Kleene.
- Ex.: “qu*x” matches “quuuuuuuux” and “qx”.
- Ex.: “(foo|bar)*” matches the empty string and “foobarbarfooobar”.

5.3 Escaping

- Backslashes cancel the special meaning of characters in regular expressions.
- Careful, you have to double them in Java so they make it through to the String itself.
- Ex.: “ba\(r|z\)” matches “ba(r|z)” but not “bar” or “baz”.
- You would declare that last expression like this in Java:

  ```java
  String pattern = "bar\((r|z)\";";"```
5.4 grep and egrep

- **grep** is a Unix command that uses regular expressions to match lines of text.
- **egrep** is a variant on **grep** that uses the same regular expression syntax in these notes, so I'll use it today.
- Both **grep** and **egrep** match a line if any part of the line matches the pattern.
- We'll use the special characters `^` (beginning of line) and `$` (end of line) to force our pattern to match the entire line.
- `/usr/share/dict/words` is a list of words used by spell-checkers; I'll use it for a demo.

```
$ egrep '^fo*$' /usr/share/dict/words  # Kleene star
  f
  foo
$ egrep '^.*(baze|nobite).*$' /usr/share/dict/words  # alternation
  abaze
  baze
  bombazet
  bumbaze
  cenobite
```

5.5 Exercise

You can match an integer whose digits are entirely ones and zeros with the pattern “(-1)(0|1)(0|1)*”. How would you change this to match a floating point number like “-101.110”?

5.6 Shorthand

- There are many shorthand notations in regular expression notation.
- Bracketed character classes are a shorthand for long alternations where each alternative is a single character.
  - Ex.: “[a-f0-9]” is equivalent to “(a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9)”.
- ? means “zero or one times” and + means “one or more times,” like * means zero or more times.
  - Ex.: “a?(foo)+” is equivalent to “(a|)foo(foo)*” and matches “afoo”, “foofoo”, “afoofoofoo”, etc.
5.7 Regular expressions in Java

- The `String.match` function returns a boolean indicating whether a `String` matches a regular expression.

- The `String.split` function splits a string into a `String[]`, where the elements are separated by matches to the regular expression. A `String` matches a regular expression.

- The `String.replaceAll` and `String.replaceFirst` methods replace all matches of a pattern with a replacement `String`.

- Simple examples: `RegexDemo.java`.

- Applied example: `FileRegexDemo.java`. 
Nov. 27: Graphics in Java

6.1 Warm-up: Color

- When specifying a color in software, you can sometimes specify it by name.
  - Java has `Color.GREEN`, `Color.MAGENTA`, etc.
- To refer to a color without a name, it must be broken into components.
- “RGB” (red, green, blue) is the most common.
- On a scale of zero to one, 1.00-0.00-1.00 would be a bright purple.
- Add bright red light to bright blue light, and you see purple.
- Usually written as three pairs of two hexadecimal digits, each representing an unsigned binary fraction.
- Humans can perceive about a hundred levels of any component, so more digits doesn’t help.
- “FF00FF” is the purple referenced above.

Some other colors:
  - black: 000000
  - white: FFFFFFFF
  - red: FF0000
  - green: 00FF00
  - blue: 0000FF
  - gray: 808080
  - olive green: 3C8031
  - etc.

6.2 Other color models

- “CMYK,” (cyan, magenta, yellow, black) is used for printing.
- It starts with white and specifies how much light of each type is removed instead of starting with black and specifying how much light is added.
- “HSL” (hue, saturation, lightness) and “HSV (a.k.a. HSB)” (hue, saturation, value/brightness) are sometimes easier to work with.
- “CIE 1931 XYZ” and “CIE Lab” are intended to represent human visual capabilities.
- “RGBA” is like RGB, but contains an additional component specifying transparency. (The “A” stands for “alpha.” I don’t know where that term comes from.)
6.3 Windowing toolkits and window systems

- A window system is a piece of software that displays a graphical user interface.
  - This generally refers to the aspect the user sees.
- A window toolkit is the software that lets a programmer create graphical user interfaces.
  - This generally refers to the aspect the programmer sees.
  - The distinction between a window system and a windowing toolkit is often blurry.
- Microsoft Windows started out as a window system built on top of DOS, but eventually came to refer to the entire operating system.
- Mac OS X, Windows 7, iOS, and Android each have distinct windowing systems.
- Unix and Linux have multiple windowing systems.
  - On computers you think of as “Unix” or “Linux,” X Windows, or just “X” is the most common window system.
  - Mac OS and iOS are actually Unix, and Android is actually Linux, but they all use other systems.
  - GNOME uses the GTK toolkit
  - KDE uses the Qt toolkit. (So does Symbian.)
- Graphical Java programs can run on many windowing systems, but have their own windowing toolkit, AWT/Swing.
- AWT/Swing programs can run on any supported window system.

6.4 AWT/Swing

- AWT stands for “Abstract Window Toolkit.”
- It was the toolkit that shipped with the first version of Java in 1995.
- Its classes are in the java.awt package.
- Swing is an extension of AWT, released with Java 1.2 in 1998.
- Its classes are in the javax.swing package.
- Together, AWT and Swing are the main Java windowing library.
- They are obsolete and rarely used to create programs other than tools for Java development (e.g. Eclipse, Android Development Kit).
• We want to do graphics in Java, so we’re stuck with it unless we want to do Android.

• Android requires you to either have an Android phone or run a slow emulator.

• Future terms I may use Android, but I didn’t have enough time to set up the necessary infrastructure.

• We want to do graphics, so AWT/Swing it is.

6.5 Containers, Components, and LayoutManagers

• javax.swing.JFrame is a class that represents a window on the screen.

• java.awt.Container is an abstract class that represents any graphical element that can contain other graphical elements.
  • A JFrame is a kind of Container.
  • Its setLayout and add methods are the most interesting.

• A javax.swing.JComponent is a graphical element like a button or a string of text that can be placed in a window for a user to view or interact with.
  • You can extend JComponent and override paintComponent to create a custom component.
  • Use event listeners to catch mouse actions and key presses on a custom component.
  • Or just use predefined components like JButton and JTextArea.

• See demonstration of these ideas in ColorDemo.java.

6.6 Running graphical Java programs

• If you work on a Mac locally:
  • If the java and javac tools work, or if you’re using Eclipse, you probably don’t need to do anything extra to be able to run graphical Java programs.

• If you work on i5 (or another Unix machine) from a Mac:
  • You can run X Windows programs on i5 and have their windows appear on your Mac with a program called an “X server.”
  • Download XQuartz from http://xquartz.macosforge.org and install it.
  • Then, pass the -X flag to ssh when you connect to i5:
    ssh -X netid@i5.nyu.edu
– When connected to i5, you can see if the process worked by typing “xlogo &”. You should see a window with this in it:

• If you work on Windows locally with Eclipse or some other IDE:
  – You probably don’t need to do anything extra.

• If you work on i5 (or another Unix machine) from Windows:
  – I’ve never done this, but it might work. Let me know how it goes for you.
  – Open Cygwin/X, then open PuTTY.
  – In the “Connection > SSH > X11” pane of the window PuTTY opens when first started, check “Enable X11 forwarding.”
  – Connect to i5, then type “xlogo &” to test it. You should see a window that looks like the one above.
7  Nov. 29: More graphics

7.1 Components
- A component is anything that can be displayed on the screen.
- `JComponent` is the parent class of most non-window components you will use.
- Override `paintComponent` to create your own.
- Call `repaint` on a component if some underlying state changes and a component needs to be redrawn.
- Painting, repainting, and invalidating is complicated; see Painting in AWT and Swing for details.

7.2 Commonly used JComponents
- `JPanel` – a container for other components
  - see also this tutorial.
  - discussed in Nested panels section
- `JButton`: standard push button
- `JLabel`: static text
- `JTextField`: editable line of text
- `JTextArea`: multiple editable lines of text
- `JScrollBar`: a scrollbar; see also JScrollPane

7.3 LayoutManagers
- You can add a sequence of components to a JFrame or JPanel, but you rarely want to specify exactly where in pixel coordinates.
- What if your program is translated to another language and the text strings change length?
- `LayoutManager` is an abstraction that lets you specify how you want to lay out a container conceptually without specifying exact coordinates.
- You can set a `LayoutManager` with the `setLayout` method of JFrame or JPanel.
7.4 *FlowLayout*
- Fills up rows of components sequentially, as if they were words filling up a page
- Can change to right alignment, analogously to right-to-left languages like Hebrew
- In practice, looks ugly unless you only have one row
- Degrades nicely when a window is oddly shaped.
- See demo at LayoutDemo.java.

7.5 *GridLayout*
- Lays out components in a grid, with each cell of the grid the same size.
- Also often ugly on its own.
- See demo at LayoutDemo.java.

7.6 *BorderLayout*
- Lays out up to five components with one in the center, one to each side, and one above and below.
- With fewer than five, the space for the missing components is filled by the ones provided.
- Also often ugly on its own.
- See demo at LayoutDemo.java.

7.7 *Nested panels*
- You rarely want a layout exactly like what the layout managers provide.
- You can nest them with JPanel.
- See demo at PanelDemo.java.
- In PanelDemo.java, a BorderLayout with a JTextArea is in the CENTER position and a JPanel is in the SOUTH position.
- The JPanel has a FlowLayout with TRAILING alignment and contains two JButton components.
7.8 Life demo

https://subversive.cims.nyu.edu/csci0101/demo/life
8 Dec. 4: Events and inner classes

8.1 Events

- In most graphical toolkits, AWT/Swing included, user actions are handled by a “listener.”
- Details vary toolkit to toolkit, but the concept remains the same.
- Some chunk of code says requests that a particular method be called whenever the user performs a particular action.
- Common AWT interfaces that a listener might implement are:
  - `MouseListener`, for mouse clicks
  - `KeyListener` for keyboard presses
  - `ActionListener` for actions taken on components like button clicks
- Listeners have a method for each type of event they can handle, like `mousePressed`, `mouseReleased`, etc. for `MouseListener`.
- For interfaces with many event types, you can extend an abstract class like `MouseAdapter`.
  - `MouseAdapter` implements every `MouseListener`, `MouseMotionListener`, and `MouseWheelListener` method with a do-nothing method.
  - You only override the methods for the events you care about.
- See demo: `EventDemo.java`

8.2 Inner classes

- You can define a class inside another class.
- The inner class can reference methods and fields of the object that created it, even private ones.
- If an inner class is declared `static`, it can only reference static fields and methods of the class of the object that created it.
- Static methods can only create objects of an inner class if the inner class is static.
- Otherwise, which object would the inner class be referring to if it referenced a member of the enclosing class?
- See demo: `EventDemoInner.java`
8.3 Anonymous inner classes

- Often, you just want to do something small when an event happens.
- Creating a whole new class seems excessive.
- The code for the event handler is far from the code that installs it, probably in a different file.
- Anonymous inner classes let you define a class right where you need it.
- Anonymous inner classes only be instantiuated where they are defined.
- Creates a JComponent which is a solid red oval:

```java
JComponent redOval = new JPanel() {
    @Override protected void paintComponent(Graphics g) {
        g.setColor(Color.RED)
        g.fillOval(0, 0, getWidth(), getHeight());
    }
};
```

- A more common use case (commonly found inside constructors of classes that extend JComponent:

```java
addMouseListener(new MouseAdapter() {
    @Override public void mouseReleased(MouseEvent e) {
        changeSomeState(e.getX(), e.getY());
        repaint();
    }
});
```

- This way, the code that processes the event is inside the component, not scattered in several different event handler files.
- Often, it is cleaner to separate the user interface and underlying logic.
- When you do this, event handlers will often be two lines of code: “change state” and “repaint component.”
- See demo: EventDemoAnon.java
9 Dec. 6: Networking

9.1 How do computers talk to each other?

- A network is a collection of computers that can talk to each other over wires, radio, or other means.
- The internet is collection of connected networks run by different organizations.
- Networking protocols specify the conventions used by different devices on a network that wish to communicate with each other.

9.2 Protocol layers

- Networking protocols are specified at multiple levels, each wrapping the higher-level protocol.
- Physical: the voltages, connector types, etc. corresponding to the physical devices (Ethernet, Wi-Fi)
- Link: the conventions used for two directly-connected devices that wish to talk to each other (Ethernet, Wi-Fi)
- Internet: the conventions used for two indirectly-connected devices that wish to talk to each other (IP, IPv6)
- Transport: the conventions used for two applications that wish to talk to each other over a network reliably and without conflicting with other applications (TCP, UDP)
- Application: the conventions that apply to particular type of application communication (DNS, HTTP for downloading web pages, SMTP for sending email)

9.3 Internet addresses

- We’re mostly going to talk about the internet layer and higher layers.
- Every computer that can talk to other computers over IP has a four byte address, usually written in the form “18.72.0.3,” where each part is a number in the range 0 to 255.
- There are only four billion such numbers, and we ran out, so we’re moving to IPv6, which has 128-bit addresses, usually written in the form “2001:0db8:85a3:0042:0000:8a2e:0370:7334.”
- You sometimes also see 48-bit hex addresses like “08:00:07:A9:B2:FC.” Those are “MAC addresses” or “Ethernet addresses” used at the link layer, not the internet layer.
• Link-layer addresses are fixed to a particular computer, but internet addresses change when you move to a new network.

• Otherwise, packets from computers not directly connected to you would not be able to find you.

• A protocol called DNS maps a name like “www.nyu.edu” to an internet address.

• You can do a DNS lookup yourself with the host command on Mac OS or i5:

$ host www.nyu.edu
www.nyu.edu is an alias for WEB.nyu.edu.
WEB.nyu.edu has address 128.122.119.202
WEB.nyu.edu has IPv6 address 2607:f600:8002:1::7
WEB.nyu.edu mail is handled by 20 MX.nyu.edu.

9.4 Transport-layer (TCP/UDP) addresses

• At the internet level, computers send packets of data to other computers.

• At the transport level, programs send continuous streams of data to other programs.

• At the transport level, addresses are the combination of an internet address and a port.

• A port is a 16-bit unsigned integer specifying which program on a computer is sending and receiving the messages.

• Servers listen on well-known ports.
  
  – The web server on a computer traditionally listens on port 80.
  – You sometimes see ports in URLs, e.g. www.example.com:8080 means “look up www.example.com in DNS, then connect to the machine with that internet address on port 80.”

9.5 Application layer

• Physical, link, internet, and transport layers are standardized.

• IPv4, in particular, has been in need of replacement for twenty years and hasn’t been fixed because the standard is so widely implemented.

• Application layers, on the other hand, can be created and modified at will.

• This is a strong point of the internet: the “end-to-end” property.
The internet takes care of getting data from point A to point B, with all of the interesting logic on the ends.

It doesn’t matter whether the intermediate link is a satellite or a fiber optic cable, who owns it, or how many individual packets are lost due to sunspots; it all looks the same to the program.

In URLs, you can identify the application layer protocol from the prefix like “http://”.

9.6 Talking to a server by hand

A simple application protocol, “daytime,” does not have any client data at all. A client connects, the server sends it the time, and the connection ends. If a computer is running a daytime server, it listens on port 13. (You can see all the well-known ports in /etc/services). You can talk to a server interactively and do the application-layer protocol yourself with the nc (netcat) command.

```
$ nc time.mit.edu 13
Sat Nov 10 21:51:33 2012
```

For a more complicated example, you can speak HTTP, the protocol for downloading web pages, yourself:

```
(you) $ nc www.nyu.edu 80
(you) GET / HTTP/1.1
(you) Host: www.nyu.edu
(you)
(server) HTTP/1.1 200 OK
(server) Date: Sun, 11 Nov 2012 02:27:00 GMT
(server) Content-Length: 28903
(server) Content-Type: text/html
(server) ...
(server) <title>New York University</title>
(server) ...
```

9.7 Networking in Java

- The InetAddress class represents an internet-layer address.
- The SocketAddress class represents a transport-layer address.
- The Socket class represents a transport-layer connection.
- The ServerSocket class allows a server to listen for connections on a port. When a client connects to that port, ServerSocket gives back a Socket over which the client and server can communicate.
- Demonstration in TimeClient.java and TimeServer.java.
10 Dec. 11: Concurrency

10.1 Walking and chewing gum at the same time

- *Concurrency* refers to doing more than one thing at once.
- In a computer with more than one processor, more than one program/function/method may be executing at a given time.
- Even if there is only one processor, different bits of code can appear to run simultaneously by taking turns (or by the operating system forcing them to take turns).
- Even if two different chunks of code can’t take turns and there is only one processor, input/output devices like disks and network adapters can be doing things while the main processor runs a program.
- More going on at once means more gets done.
- Especially improves responsiveness as perceived by users.

10.2 Threading

- One way to have concurrency is to run multiple programs at once.
- Another way is to have a single program do multiple things at once.
- In a concurrent program, a *thread* is one of the things the program is doing.
- Each thread gets its own local variables and its own call stack.
- But, objects can be referred to by multiple threads if multiple threads can get a reference to the same object.
- A *task*, as used in this class, refers to something a thread might do.
- Threads can run only one task at once, but they might run several tasks, one after another.

10.3 Threading in Java

- A thread in java is represented by the `Thread` class.
- A task is represented by the `Runnable` interface.
- The simplest way to write a concurrent program in java is to create `Runnable` objects for each thing you want the program to do, then create `Thread` objects for each of them, then start the threads.
- The threads will run concurrently.
• If any of them has to wait for IO, network, or user input, the rest will continue independently.

• If you have four processors, as long as at least four threads are not waiting for something external, all four will be in use at all times.

• See ConcurrentTimeServer.java.

10.4 Race conditions

• A race condition occurs when program behavior depends on timing in a way that makes its behavior unpredictable.

• See BadCountingTimeServer.java.

• Tasks in BadCountingTimeServer use taskCount, a variable that can change while the task runs.

• Not even taskCount++ is safe.

• To increment a variable, you need to read it, increment it, and write it back out to memory.

• Two tasks can read the variable at the same time, increment it, and write it back out.

• Then, the variable will only have been incremented by one.

10.5 Locking

• A lock is used to coordinate multiple threads.

• A lock can be held by at most one thread at once.

• Each shared object is guarded by a lock.

• To read or write the shared object, you acquire the lock.

• When you are done with the shared object, you release the lock.

• If a thread tries to acquire a lock another thread has already acquired, it waits (“blocks”) until the lock is released.

• If two objects only make sense when modified together, they should be guarded by the same lock.

  – Example: bank account balances: acquire lock, increase one, decrease other, release lock.
10.6 Locking in Java

- Any object can be used as a lock in Java.
- To acquire a lock and run some code with that lock held, put the code in a `synchronized` block.

```java
// ...code run without lock...
...
// Next line blocks until lock on lockObject is acquired.
synchronized (lockObject) {
    // ...code to run with lock...
} // Lock released here.
...
// ...code run without lock...
```

- See `CountingTimeServer.java`.

10.7 Issues

- Concurrency is hard.
- Locking does not solve all problems.
- Locking limits concurrency when multiple threads all need access to the same object at the same time.
- Worse, a program may deadlock if two threads need locks held by each other to proceed.
- Concurrency bugs are often non-deterministic and hard to identify and fix.

10.8 Optional readings

- Oracle Java Concurrency Tutorial: The minimum you need to know to write concurrent Java code.
- Java Concurrency in Practice, by Brian Goetz et al: More than you need to know, but quite good, and a pleasure to read even if you don’t want to write concurrent Java code.