CSSE 220 Day 19

Hardy Solution

Compiling/Running Programs from the Command Line

Vector Graphics Assignment

Questions?

Exam 2: Thursday, Feb 5

- Same rules and format as Exam 1
- Through Chapter 14 (approximately) in the textbook

Hardy Redux

- Organization
  Guaranteeing we don't miss any
  A major speedup

These Slides are on ANGEL in the Lessons→Solutions folder

Command line

- Compiling and Running a Java Program

Why run from the command line?

- User may not have Eclipse
  - or may not want to learn to use it
- We can have our program do different things based on the arguments.
- Perhaps our Java program is one element of a larger script
- Commands:
  - javac – compile Java class(es)
  - java – run a Java class (must contain main() method)
  - javadoc – generate HTML from javadoc

Example (expects one command-line arg)

```
public class Factorial{

   public static BigInteger factorial(int n) {
      if (n < 0)
         throw new IllegalArgumentException("" + n);
      BigInteger prod = BigInteger.ONE;
      for (int i = 1; i <= n; i++)
         prod = prod.multiply(new BigInteger(i + ""));
      return prod;
   }

   // Calculates the factorial of its command-line arguments
   // @param args array of strings: command-line arguments.
   public static void main(String[] args) {
      try {
         int n = Integer.parseInt(args[0]);
         System.out.println(n + "! = " + factorial(n));
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Command-line argument required");
      } catch (NumberFormatException e) {
         System.out.println("Argument must be an integer");
      } catch (IllegalArgumentException e) {
         System.out.println("Factorial arg cannot be negative: " +
                                            e.getMessage());
      }
   }
}
```

To Do:

- Start Menu → Run → cmd → Enter key
- CD to your Eclipse project folder, then to Factorial/src
- dir
- javac Factorial.java
- dir
- java Factorial
- java Factorial xyz
- java Factorial -5
- java Factorial 75

You can also do **javac *.java** to compile all Java source files in a folder.

How do we tell Eclipse about command-line arguments for testing purposes

Vector Graphics Assignment

- A team project to create a scalable graphics program.

Goals of Vector Graphics Project

- Express your creativity
  - There are few constraints on the layout or the user interaction.
- Dig into documentation to investigate the various Java Swing classes that are available
- Hone your teamwork skills
- Experience development cycles

Getting information on Swing

- Resources
  - The Java API documentation
    - http://java.sun.com/javase/6/docs/api/
    - In the list of packages, scroll down to `javax.swing` ; Also see `javax.swing.*`
  - the *Java Swing Tutorial*
  - *Java Swing* book
    - For access, see the course syllabus. Look for

      How to Access Safari Tech Books Online
  - VectorGraphics discussion forum on ANGEL
  - etc. Feel free to post your favorite resource links on the discussion forum

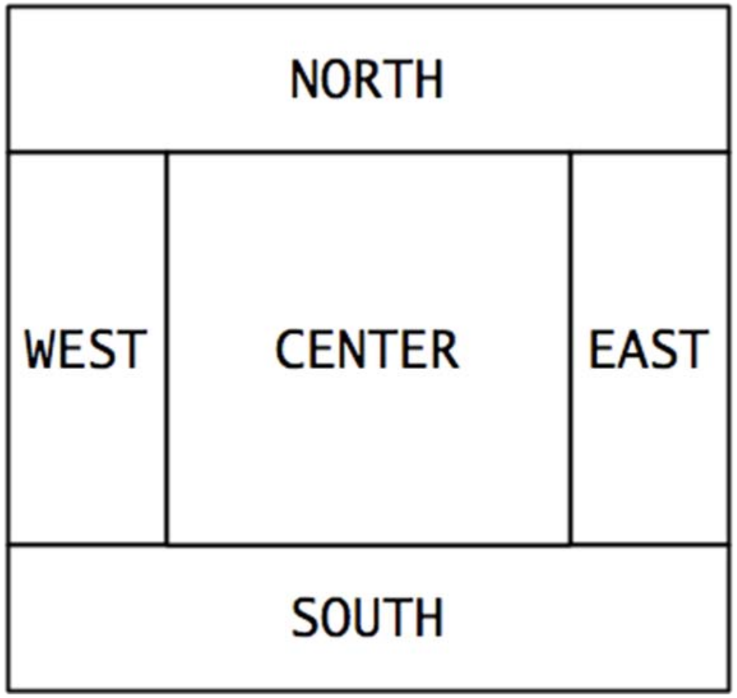
Layout Manager Review

- I placed these slides here for reference, we discussed them earlier

Recall: How many components can a JFrame show by default?

- Answer: 5
- We use the two-argument version of **add**:

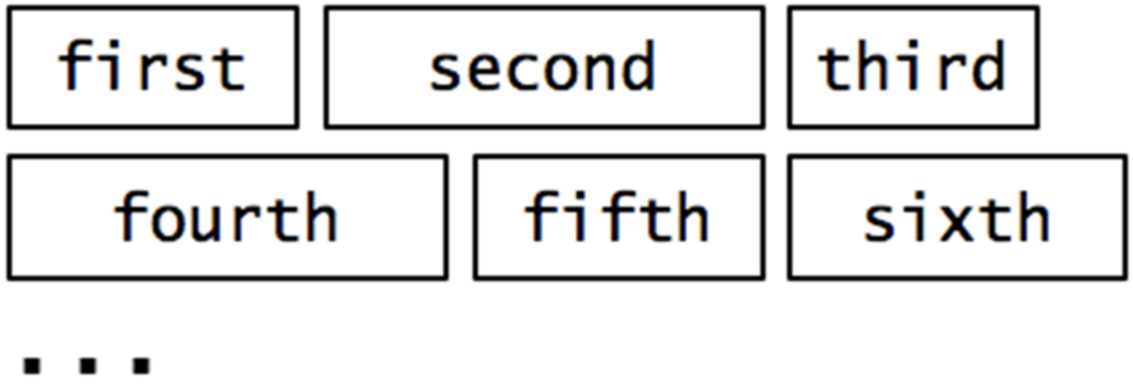
-
  ```
  JPanel p = new JPanel();
  frame.add(p, BorderLayout.SOUTH);
  ```
- **JFrame**'s default **LayoutManager** is a **BorderLayout**
- **LayoutManager** instances tell the Java library how to arrange components
- **BorderLayout** uses up to five components



Recall: How many components can a JPanel show by default?

- Answer: arbitrarily many
- Additional components are added in a line

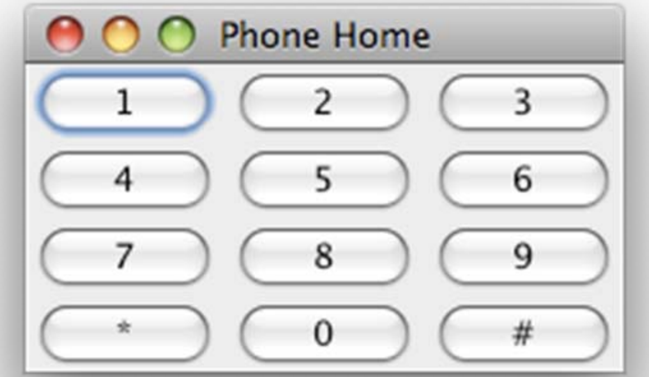
- **JPanel**'s default **LayoutManager** is a **FlowLayout**



Setting the Layout Manager

- We can set the layout manager of a JPanel manually if we don't like the default:

```
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(4,3));
panel.add(new JButton("1"));
panel.add(new JButton("2"));
panel.add(new JButton("3"));
panel.add(new JButton("4"));
// ...
panel.add(new JButton("0"));
panel.add(new JButton("#"));
frame.add(panel);
```



Lots of Layout Managers

- A **LayoutManager** determines how components are laid out within a container
  - **BorderLayout**. When adding a component, you specify center, north, south, east, or west for its location. (Default for a JFrame.)
  - **FlowLayout**: Components are placed left to right. When a row is filled, start a new one. (Default for a JPanel.)
  - **GridLayout**. All components same size, placed into a 2D grid.
  - Many others are available, including **BoxLayout**, **CardLayout**, **GridBagLayout**, **GroupLayout**
  - If you use the **null** for the **LayoutManager**, then you must specify every location using coordinates
    - More control, but it doesn't resize automatically

Additional Resources on Layout Managers

- Chapter 18 of Big Java
- Swing Tutorial
  - http://java.sun.com/docs/books/tutorial/ui/index.html
  - Also linked from schedule

Java Swing book in Safari Books online (see the course syllabus)

Vector Graphics Design

- Verify SVN repository, check-out project
- Exchange contact information
- Begin work on first milestone (see HW 19)