CSSE 220 Day 12

Object-Oriented Design
File I/O
Exceptions

Checkout the *FilesAndExceptions* project

Questions?

Please complete the Project Team Preference Survey

Today's Plan

- LayoutManagers for Java GUIs
- BallWorlds work time

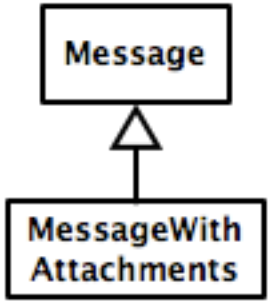
Describe the Relationships

- Classes usually are related to their collaborators
- Draw a UML class diagram showing how
- Common relationships:
  - **Inheritance**: only when subclass **is a** special case
  - **Aggregation**: when one class **has a** **field** that references another class
  - **Dependency**: like aggregation but transient, usually for method parameters, **"has a" temporarily**
  - **Association**: any other relationship, can label the arrow, e.g., **constructs**

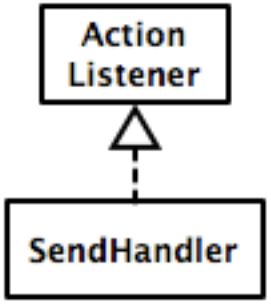
NEW!

Summary of UML Class Diagram Arrows

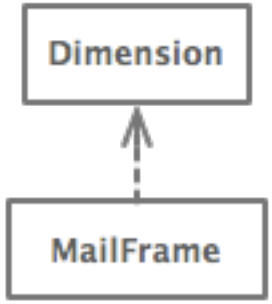
Inheritance (is a)



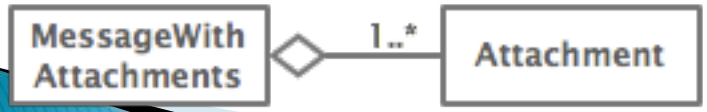
Interface Implementation (is a)



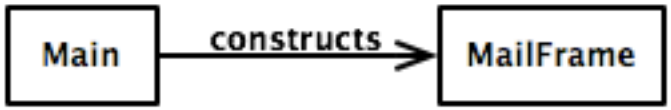
Dependency (depends on)



Aggregation (has a)



Association



Q1

Object-Oriented Design

- Draw UML class diagrams based on your CRC cards
- Initially just show classes (not insides of each)
- Add insides for two classes

Some Notes on Layout Managers

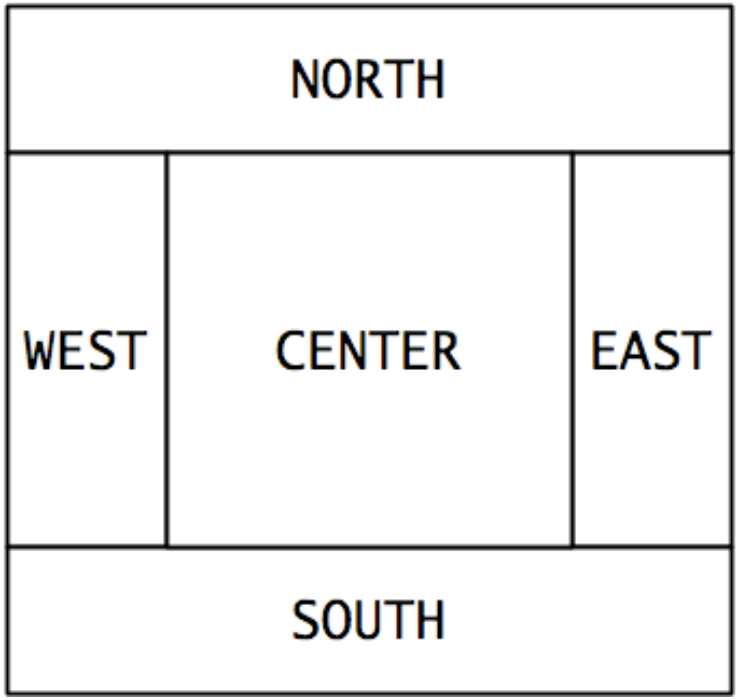
- When JFrame's and JPanel's defaults just don't cut it.

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

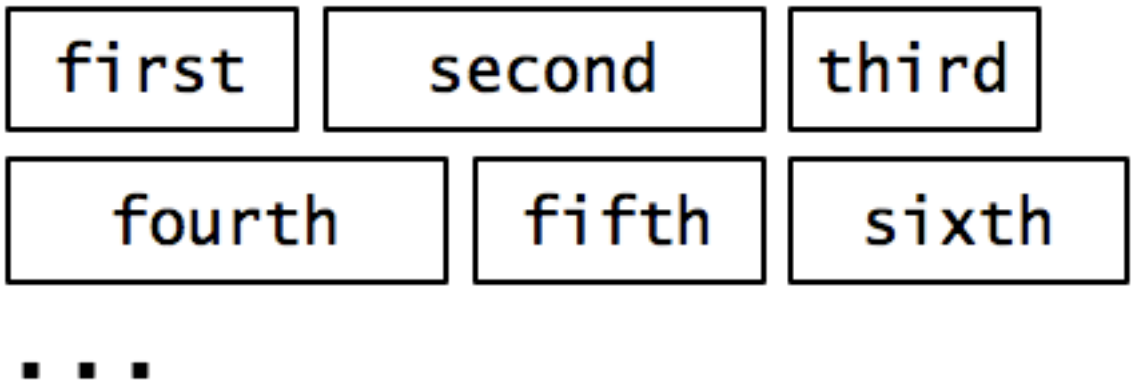


Q2
  ```

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 `null` for the **LayoutManager**, then you must specify every location using coordinates
    - More control, but it doesn't resize automatically

Q3

Additional Resources on Layout Managers

- Chapter 18 of Big Java

- Swing Tutorial
  - http://docs.oracle.com/javase/tutorial/ui/index.html
  - Also linked from schedule

Files and Exceptions

- Reading & writing files
  When the unexpected happens

Review of Anonymous Classes

- Look at GameOfLifeWithIO
  - GameOfLife constructor has 2 listeners, two *local anonymous* class
  - ButtonPanel constructor has 3 listeners which are *local anonymous* classes

- Feel free to use as examples for your project

File I/O: Key Pieces

- Input: `File` and `Scanner`
- Output: `PrintWriter` and `println`
- Be kind to your OS: `close()` all files
- Letting users choose: `JFileChooser` and `File`
- Expect the unexpected: `Exception` handling
- Refer to examples when you need to…

Q4–Q6

Exceptions

- Used to signal that something went wrong:
  - `throw new EOFException("Missing column");`
- Can be **caught** by **exception handler**
  - Recovers from error
  - Or exits gracefully

Q7

A Checkered Past

- Java has two sorts of exceptions
- **Checked exceptions**: compiler checks that calling code isn't ignoring the problem
  - Used for **expected** problems
- **Unchecked exceptions**: compiler lets us ignore these if we want
  - Used for **fatal** or **avoidable** problems
  - Are subclasses of `RunTimeException` or `Error`

Q8-Q9

A Tale of Two Choices

- Dealing with checked exceptions
  - Can **propagate** the exception
    - Just declare that our method will pass any exceptions along
    - `public void loadGameState() throws IOException`
    - Used when our code isn't able to rectify the problem
  - Can **handle** the exception
    - Used when our code can rectify the problem

Q10

Handling Exceptions

- Use try-catch statement:
  - ```
    try {
        // potentially "exceptional" code
    } catch (ExceptionType var) {
        // handle exception
    }
    ```
- Related, try-finally for clean up:
  - ```
    try {
        // code that requires "clean up"
    } finally {
        // runs even if exception occurred
    }
    ```

Can repeat this part for as many different exception types as you need.

LoadRunner Assignment

- Demonstrate the program

Teaming

- A team assignment
  - So some division of labor is appropriate (indeed, necessary)
- A learning experience, so:
  - Rule 1: *every* team member must participate in *every* major activity.
    - E.g., you are not allowed to have someone do graphics but no coding,
  - Rule 2: Everything that you submit for this project should be understood by *all* team members.
    - Not necessarily all the details, but all the basic ideas

Work time now

- Read the specification if you haven't done so
- Start working on your milestone 0 **due next class**
  - Try to get it done in class today so you can:
    - Get some feedback in class before it’s graded.

Plan, then do

- There are milestones due most class days:
- For next class:
  - User stories
  - CRC cards
  - UML class diagram
  - See the project description for details

  - Suggestion:
    - Plan to implement a considerable amount of functionality in Cycle 1
    - It is the longest cycle that you will have

Work Time

- BallWorlds

Q11–Q12