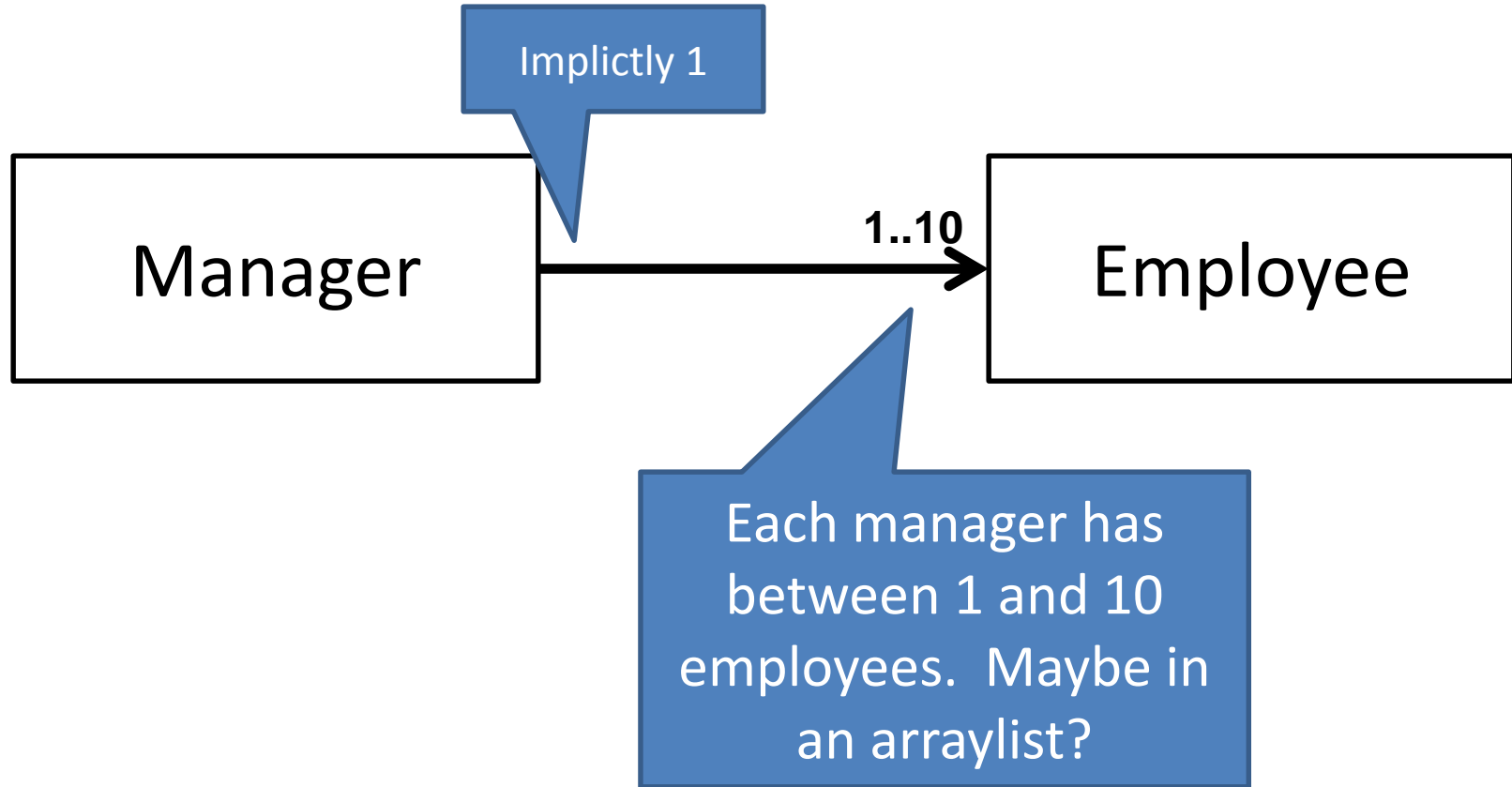


# CSSE 220

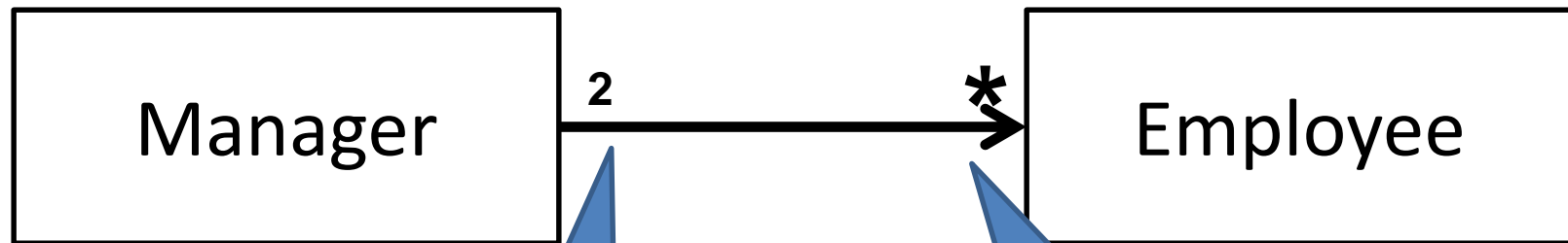
Object-Oriented Design  
Files & Exceptions

Check out *FilesAndExceptions* from SVN

# New UML Notation: Cardinality



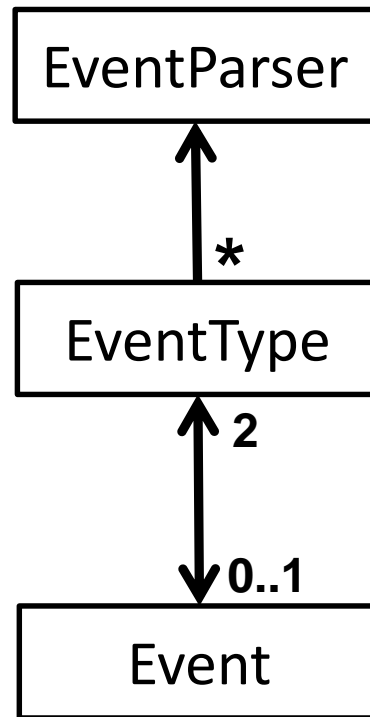
# More Cardinality



Every employee has exactly 2 managers. Note that this can be used even if there is no reference from Employee to Manager

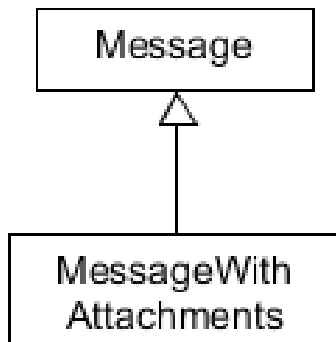
Managers have any number of employees.  
The \* means “zero to infinity” – any arbitrary number. You can also occasionally see something like 4..\* to mean 4 or more.

# What does this diagram mean?

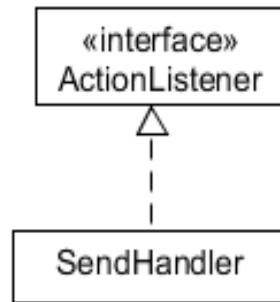


# Summary of UML Class Diagram Arrows

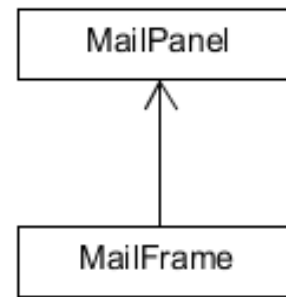
Inheritance  
(is-a)



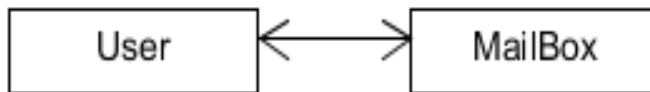
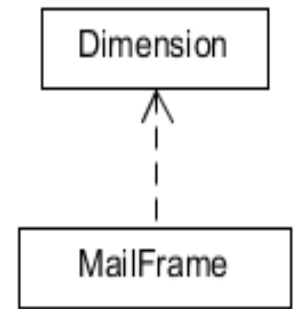
Interface  
Implementation  
(is-a)



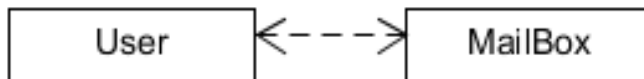
Association  
(has-a-field)



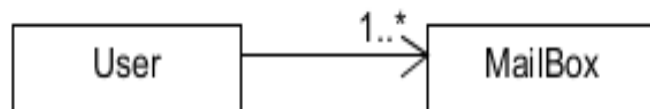
Dependency  
(depends-on)



Two-way Association



Two-Way Dependency



Cardinality  
(one-to-one, one-to-many)  
One-to-many is shown on left

Reading & writing files  
When the unexpected happens

# **FILES AND EXCEPTIONS**

# 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...

# Exception – What, When, Why, How?

- What:
  - Used to signal that something in the code has gone wrong
- When:
  - An error has occurred that cannot be handled in the current code
- Why:
  - Breaks the execution flow and passes exception up the stack



# Exception – How?

- Throwing an exception:

```
throw new EOFException("Missing column");
```

- Handling (catching) an exception:

```
try {  
    //code that could throw an exception  
}  
catch (ExceptionType ex) {  
    //code to handle exception  
}
```

- When caught you can:

- Recover from the error OR exit gracefully

# What happens when no exception is thrown?

```
Scanner inScanner;
```

```
try {
```

```
    inScanner =
```

If this line is successful

```
        new Scanner(new File("test.txt"));
```

```
    //code for reading lines
```

Code continues on

```
} catch (IOException ex) {
```

```
    JOptionPane.
```

The catch never executes

```
        showMessageDialog("File not found.");
```

```
} finally {
```

```
    inScanner.close();
```

This runs after code in try completes

```
}
```

# What happens when exception is thrown?

```
Scanner inScanner;  
try {
```

```
    inScanner =
```

If this line throws exception

```
        new Scanner(new File("test.txt"));
```

```
    //code for reading lines
```

Code after exception never executes

```
} catch (IOException ex) {
```

```
    JOptionPane.
```

This is the next line executed

```
        showMessageDialog("File not found.");
```

```
} finally {
```

```
    inScanner.close();
```

After catch is executed, this runs

```
}
```

# When exception is not handled?

```
public String readData(String filename)  
    throws IOException {
```

```
    Scanner inScanner =
```

```
        new Scanner(new File(filename));
```

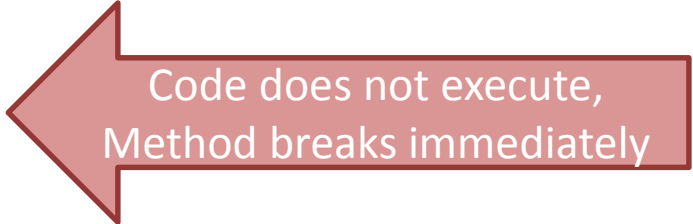
```
    //code for reading lines
```

```
    inScanner.close();
```

```
}
```



If this line throws exception



Code does not execute,  
Method breaks immediately

main -> readAllFiles -> readData

If unhandled, exception bounces to  
method that called it, then up the chain.

# A Checkered Past

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

# A Tale of Two Choices

## Dealing with **checked** exceptions

### 1. 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

### 1. Can **handle** the exception

- Used when our code can rectify the problem

# Handling Exceptions

- Use try-catch statement:

```
try {  
    // potentially “exceptional” code  
} catch (ExceptionType var) {  
    // handle exception  
}
```

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

Related, try-finally for clean up:

```
try {  
    // code that requires “clean up”  
} finally {  
    // runs even if exception occurred  
}
```

# Exception Activity

- Look at the code in FileAverage, focusing on the use of exceptions
- Solve the problems in FileBestScore



# Exam 2

- Paper part includes:
- 2 Design Problems (20 points)
- Questions about coupling & cohesion (5 points)
- Question about exceptions (4 points)
- Compile/runtime/printing question (11 points)
- Tracing a recursive function (12 points)
- **You can bring 1 sheet of notes + OO Principles for 220 + UML Cheatsheet**

# Exam 2

- Computer part includes:
- Recursion
- Problem where you must use inheritance or interfaces to remove code duplication
- Problem where you have to layout a GUI and handle updates using listeners