

Tutorial 10

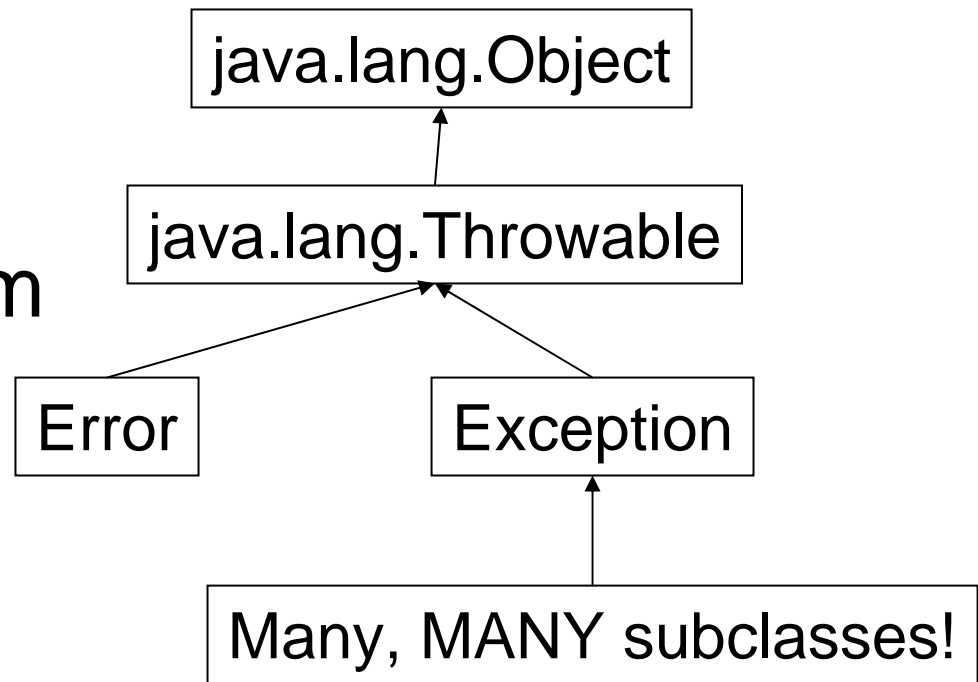
November 14 & 15, 2005

Today's Topics

- Exceptions
- Problem Set 8 – Sudoku
- Quizzes returned / questions

Exceptions

- Exceptions are objects, just like everything else in Java.
- “are objects”
literally means
they inherit from
Object.



What about Errors?

- Java makes a distinction between Errors and Exceptions.
- Both can be *thrown* (because they extend Throwable) when something unexpected happens.
- Exceptions should be *caught*, but most reasonable applications do not catch Errors.
- We will be focusing on Exceptions for this reason.
- Don't be confused – even if we say there's a program error, we really mean that an Exception was thrown, not an Error.

Exceptions – checked vs. unchecked

- The distinction here is somewhat arbitrary.
- Overall, you must check for checked exceptions (using a try/catch block) and don't have to check for unchecked exceptions.
- Another way to think of this is that checked exceptions are anticipated exceptions – you know there's a good chance one will be thrown, so you check for it.
- Example: opening a user-specified file. If they mistype the filename, you'll get a `FileNotFoundException`.

Dealing with checked Exceptions

- Option 1: try/catch block

```
public void openFile1() {  
    try {  
        // this call might throw an IOException...  
        FileInputStream f = new FileInputStream("test.java");  
    }  
    catch (IOException e) {  
        // ...so we catch it here  
    }  
}
```

- Catch either the Exception that might be thrown, or a superclass of that Exception

Dealing with checked Exceptions

- Option 2: throw the exception to the next method up the call stack: throws

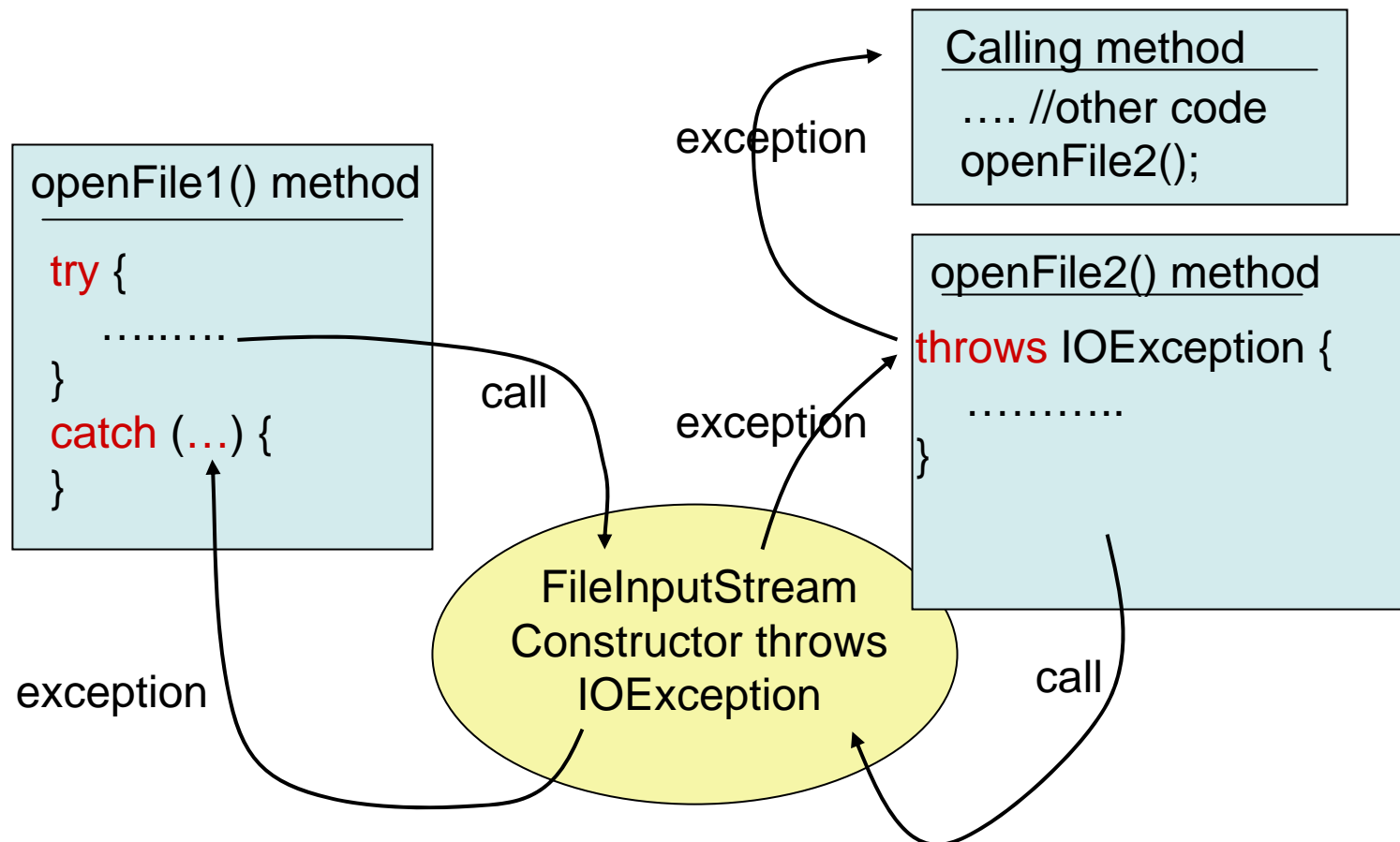
```
public void openFile2() throws IOException {  
    // this call might throw a IOException  
    FileInputStream f = new FileInputStream("test.java");  
}
```

- Since this method doesn't catch the exception, one of the methods that called it must.

Dealing with checked Exceptions

- Option 1: try/catch

- Option 2: throws



Throwing Exceptions

- Example: you write a `sqrt()` method that returns the square root of a *positive* number.
- If the argument is negative, you want to let yourself (the programmer) know by throwing an exception.

```
public double sqrt(double n) {  
    if (n < 0)  
        throw new IllegalArgumentException();  
    return Math.pow(n, .5);  
}
```

- Does this generate a checked, or unchecked exception?

Creating your own Exceptions

- To create your own type of exceptions (e.g. `TIVOException`, `TiredException`, `NoSolutionException`, etc.) just extend another exception:

```
public class DataFormatException
    extends java.lang.Exception {

    public DataFormatException()
        { super(); }

    public DataFormatException(String s)
        { super(s); }
}
```

Exceptions – the VM Stack Trace

- Does this look familiar?

```
Exception in thread "main"  
    java.lang.NullPointerException  
at MyClass.myMethod(MyClass.java:13)  
at MyClass.main(MyClass.java:8)
```

- This is called a stack trace, and shows you exactly where an Exception was created, and who was calling the method that caused the exception. Use this information to your advantage!

PS8 – Soduko/Sudoku

9 squares

9 columns

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	1	6	7	4	8	9	5
8	7	5	9	1	2	3	6	4
6	9	4	5	3	8	2	1	7
3	1	7	2	6	5	9	4	8
5	4	2	8	9	7	6	3	1
9	6	8	3	4	1	5	7	2

9 rows

PS8 – Soduko/Sudoku

- Your task: *design* a program that...
 - Allows users to create their own “boards” and save them.
 - Allows users to load previously created boards.
 - Allows a user to play a board, indicating when a rule is broken.
 - Can SOLVE a board (!!)
 - or report that it has no solution.

PS8 – Soduko/Sudoku

- In particular, this means designing:
 - A GUI
 - Use scenarios (what should happen when X occurs?)
 - Java Class skeletons (methods, data members)
- This does NOT mean:
 - Coding a GUI
 - Coding an algorithm
 - Coding much of anything at all

PS8 – Soduko/Sudoku

- Tips for the design:
 - Break it into smaller, more manageable pieces. For example:
 - ***Keep the visual display (“view”) separate from the underlying board “model” that the rules and solver actually work with.***
 - For example, don’t try to write methods that directly manipulate arrays of JTextFields to check moves.

PS8 – Soduko/Sudoku

- Another example of “breaking it down into smaller pieces”: checking to see if a given board is legal.
- What are the sub-parts of this problem?
 - Checking to see if a square is legal
 - Checking to see if a row is legal
 - Checking to see if a column is legal
- Rather than writing one monstrous method, break it into smaller steps that seem easier to accomplish, and make each step its own method.

PS8 – Soduko/Sudoku

- Hand in:
- GUI design (sketches, explanations)
- Class skeletons
 - **COMMENT!!!**
 - **You will be graded on your design, so if you do not explain the purpose of a part of a class, we won't be able to give you the points you deserve.**