Objectives

- Exceptions

New Extra Credit Opportunity

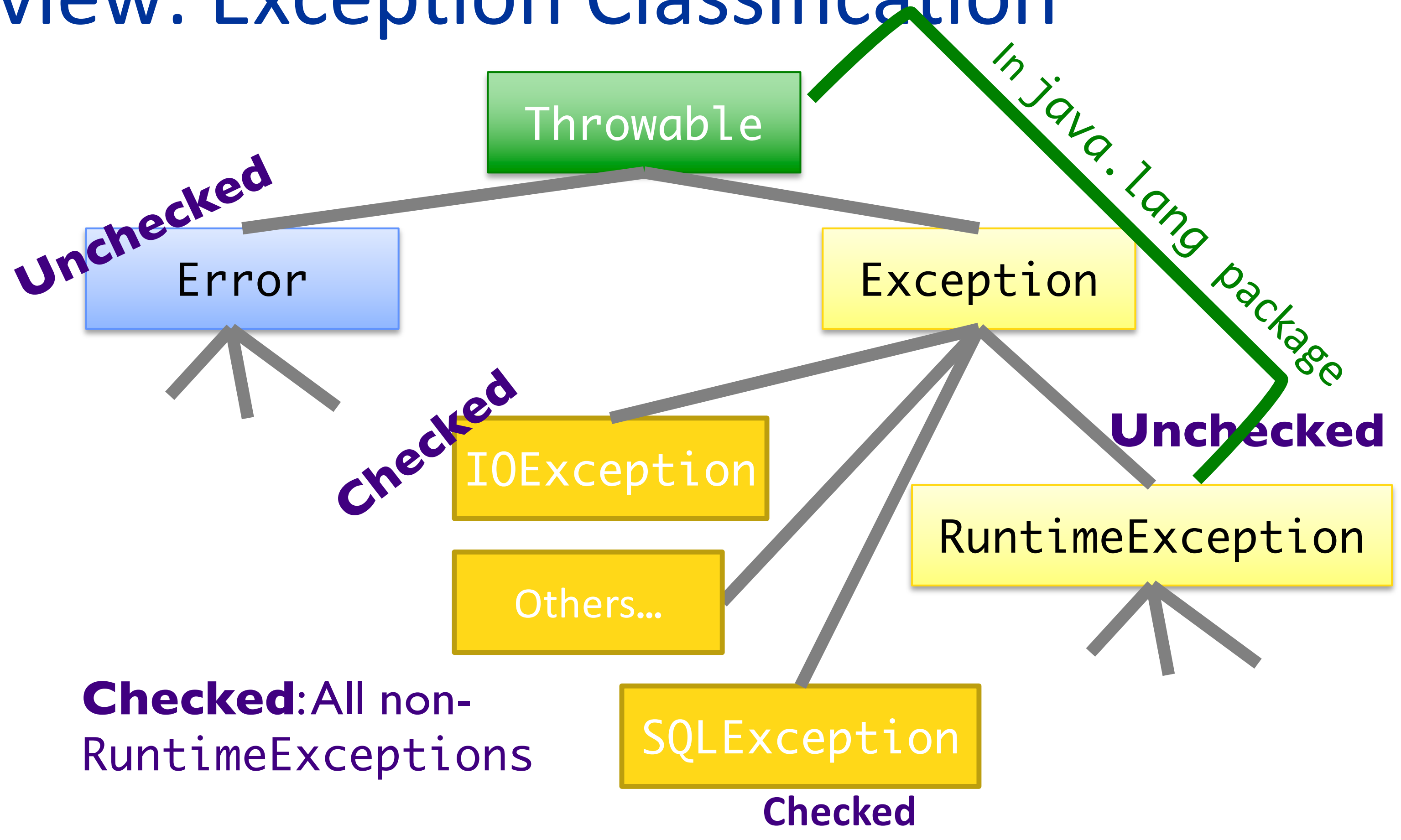
- ACM Tech Talks
- Software-engineering focused
    - Large Language Models and the End of Programming with Matt Welsh
    - Effective Developer Testing with Mauricio Aniche
    - Tradeoffs in the Software Workflow with Titus Winters
    - And more

https://learning.acm.org/techtalks-archive

Review

1. What are the benefits of the Collections Framework?
2. What is an Exception?
3. What are the different categories of exceptions?
   - What are examples (i.e., class names) of those categories of exceptions?
4. What is Eclipse? What can it do?

Review: Exception Classification



Checked: All non-RuntimeExceptions

THROWING EXCEPTIONS

Methods and Exceptions Example

- `BufferedReader` has method `readLine()`
  - Reads a line from a *stream*, such as a file or network connection
- Method header:

Part of Advertising

```
public String readLine() throws IOException
```

- Interpreting the header: `readLine` will
  - return a String (if everything went right)
  - throw an `IOException` (if something went wrong)

Advertising Checked Exceptions

- Advertising in Javadoc: document under what conditions each exception is thrown
  - `@throws` tag
- Examples of when your method should advertise the ***checked*** exceptions that it may throw
  - Your method calls a method that throws a checked exception
  - Your method detects an error in its processing and decides to throw an exception

Example: Passing an Exception "Up"

```
public String readData(BufferedReader in)
   throws IOException {
      String str1 = in.readLine();
      return str1;
}
```

Throws an `IOException`

- `readData` calls `readLine`, which can throw an `IOException`
- If `readLine` throws this exception to our method
  - `readData` *throws* the exception as well
  - Whoever calls `readData` will handle exception

Example: Throwing An Exception We Created

1. Create a new object of class **`IllegalArgumentException`**
   - Class derived from **`RuntimeException`**
2. `throw` it
   - Method ends at this point
   - Calling method handles exception

```
if (grade < 0 || grade > 100) {
      throw new IllegalArgumentException();
}
```

Equivalent in Python?

A More Descriptive Exception

- Four constructors for most Exception classes
  - Default (no parameters)
  - Takes a `String message`
    - Describe the condition that generated this exception more fully
  - And 2 more

```
if (grade < 0 || grade > 100) {
        throw new IllegalArgumentException(
                "Grade is not in valid range (0-100)");
}
```

The best error messages include all state that could have contributed to the problem

Common Exception Classes

Name	Purpose
`IllegalArgumentException`	When caller passes in inappropriate argument
`IllegalStateException`	Invocation is illegal because of receiving object’s state. (Ex: closing a closed window)

- Both inherit from `RuntimeException`
- May seem like these cover everything but only used for certain kinds of illegal arguments and exceptions
- Not used when
  - A null argument passed in; should be a `NullPointerException`
  - Pass in invalid index for an array; should be an `IndexOutOfBoundsException`

Goal: Failure Atomicity

- After an object throws an exception, the object should be in a well-defined, usable state
  - A failed method invocation should leave object in state prior to invocation
- Approaches:
  - Check parameters/state before performing operation(s)
  - Do the failure-prone operations first
  - Use recovery code to “rollback” state
  - Apply to temporary object first, then copy over values

Birthday Error Handling Discussion

- Design decision:
  - Since month and day are not independent, should be set *together* rather than separately
- Check all the error cases before setting the instance variables
  - Don't want an inconsistent resulting birthday
- `IllegalArgumentException` is appropriate
  - Programming error
  - Caller should catch those errors before executing program

Javadoc Guidelines about @throws

- Always report if throw ***checked*** exceptions
- Report any unchecked exceptions that the caller might reasonably want to catch
  - Exception: `NullPointerException`
  - Allows caller to handle (or not)
  - Document exceptions that are independent of the underlying implementation
- `Errors` will **not** be documented as they are unpredictable

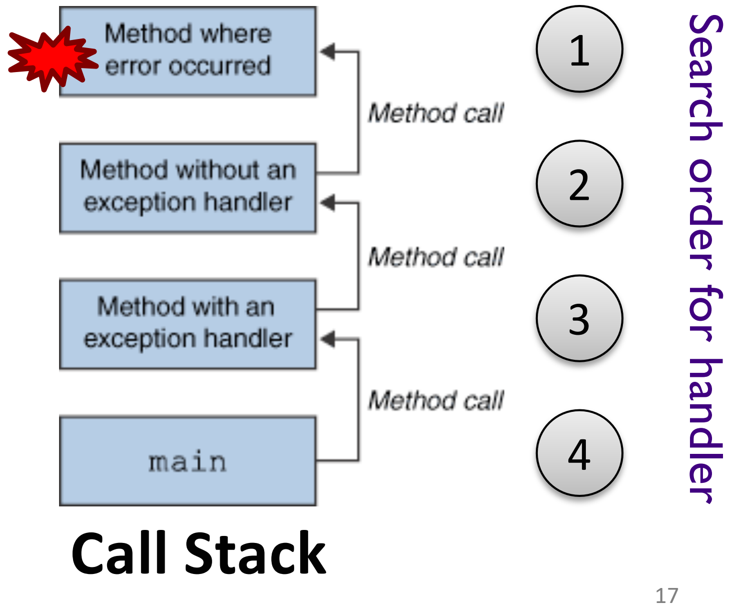
HANDLING EXCEPTIONS

Handling Exceptions

- After an exception is thrown, some part of program needs to ***catch*** it
- What does it mean to catch an exception?
  - Program knows how to deal with the situation that caused the exception
  - Handles the problem—hopefully gracefully, without exiting

Handling Exceptions

- JVM's exception-handling mechanism searches for an ***exception handler***—the error recovery code
  - Exception handler deals with a particular exception
  - Searches call stack for a method that can handle (or catch) the exception


Method where error occurred
Method call
Method without an exception handler
Method call
Method with an exception handler
Method call
main
Call Stack
1
2
3
4
Search order for handler

Try/Catch Block

- The simplest way to catch an exception
- Syntax:

Python equivalent?

```
try {
        code;
        more code;
}
catch (ExceptionType e) {
        error code for ExceptionType;
}
catch (ExceptionType2 e) {
        error code for ExceptionType2;
}
…
```

Try/Catch Block

```
try {
        code;
        more code;
}
catch (ExceptionType e) {
        error code for
        ExceptionType
}
```

- Code in `try` block runs first
- If `try` block completes without an exception, `catch` block(s) are not executed
- If `try` code generates an exception
  - A `catch` block runs
  - Remaining code in `try` block is not executed
- If an exception of a type other than `ExceptionType` is thrown inside `try` block, method exits immediately*

Try/Catch Block

```
try {
        code;
        more code;
}
catch (ExceptionType1 e) {
        error code for
        ExceptionType1
}
catch (ExceptionType2 e) {
        error code for
        ExceptionType2
}
```

- You can have more than one `catch` block
  - To handle > 1 type of exception
- If exception is not of type `ExceptionType1`, falls to `ExceptionType2`, and so forth
  - Run the first matching `catch` block

Can catch any exception with **`Exception e`** but won't have customized messages

Try/Catch Example

```
public void read(BufferedReader in) {
        try {
                boolean done = false;
                while (!done) {
                        String line=in.readLine();
                        // above could throw IOException
                        if (line == null)
                                done = true;
                }
        }
        catch (IOException ex) {
                ex.printStackTrace();
        }
}
```

Prints out stack trace to method call that caused the error

Try/Catch Example

```
public void read(BufferedReader in) {
        try {
                boolean done = false;
                while (!done) {
                        String line=in.readLine();
                        // above could throw IOException
                        if (line == null)
                                done = true;
                }
        }
        catch (IOException ex) {
                ex.printStackTrace();
        }
}
```

Alternatively, a more precise (child Exception class) `catch` may help pinpoint error
But could result in messier code

The finally Block

- Optional: add a finally block after all catch blocks
  - Code in finally block **always** runs after code in try and/or catch blocks
    - After try block finishes or, if an exception occurs, after the catch block finishes

```
try {
        …
}
catch (Exception e) {
        …
}
finally {
        …
}
```

- Allows you to clean up or do maintenance before method ends (one way or the other)
  - E.g., closing files or database connections

FinallyTest.java

Practice: `try/catch/finally` Blocks

```
try {
        statement1;
        statement2;
}
catch (EOFException e) {
        statement3;
        statement4;
}
finally {
        statement5;
}
statement6;
```

- Which statements run if:
  1. Neither `statement1` nor `statement2` throws an exception
  2. `statement1` throws an `EOFException`
  3. `statement2` throws an `EOFException`
  4. `statement1` throws an `IOException`

Practice: `try/catch/finally` Blocks

```
try {
        statement1;
        statement2;
}
catch (EOFException e) {
        statement3;
        statement4;
}
finally {
        statement5;
}
statement6;
```

- Which statements run if:
  1. Neither `statement1` nor `statement2` throws an exception
     - 1, 2, 5, 6
  2. `statement1` throws an `EOFException`
     - 1,3,4,5,6
  3. `statement2` throws an `EOFException`
     - 1,2,3,4,5,6
  4. `statement1` throws an `IOException`
     - 1,5

Fun Fact: Python also has `finally`

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

https://docs.python.org/3/tutorial/errors.html

Fun Fact: Python also has `finally`

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionErro
        print("division by 
    else:
        print("result is", 
    finally:
        print("executing fi
```

```
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 3, in divide
TypeError: unsupported operand
type(s) for /: 'str' and 'str'
```

https://docs.python.org

Catching More Than One Exception Type

- Can catch multiple exception types in one catch block

```
try {
        statement1;
        statement2;
}
catch (EOFException | SQLException e) {
        statement3;
        statement4;
}
finally {
        statement5;
}
```

What to do with a Caught Exception?

- Print/log the stack after the exception occurs

```
java.io.FileNotFoundException: fred.txt
    at java.io.FileInputStream.<init>(FileInputStream.java)
    at java.io.FileInputStream.<init>(FileInputStream.java)
    at ExTest.readMyFile(ExTest.java:19)
    at ExTest.main(ExTest.java:7)
```

How helpful is this output?
How user friendly is it?

What to do with a Caught Exception?

- Print/log the stack after the exception occurs
  - But, what else can we do?

- Generally, two options:
  1. Catch the exception and recover from it
  2. Pass exception up to whoever called it

Programming with Exceptions

- Exception handling is slow
- Group relevant code together
  - Scope of try/catch block should be small
- Use one big `try` block instead of nesting `try-catch` blocks
  - Speeds up Exception Handling
  - Otherwise, code gets too messy
- Don't ignore exceptions (e.g., `catch` block does nothing)
  - Better to pass them along to higher calls

```
try {
}
catch () {
}
try {
}
catch () {
}
```

```
try {
   try {
   }
   catch () {
   }
}
catch () {
}
```

```
try {
      …
      …
}
catch () {
}
```

Summary: Methods Throwing Exceptions

- API documentation tells you if a method can throw an checked exception
  - If so, you **must** handle it
- If your method could possibly throw an exception (by generating it or by calling another method that could), advertise it!
  - If you can't handle every error, that's OK…let whoever is calling you worry about it
  - However, they can only handle the error if you advertise the exceptions you can't deal with

Creating Custom Exception Class

- Try to reuse an existing exception
  - Match in name as well as semantics

- If you cannot find a predefined Java `Exception` class that describes your condition, implement a new `Exception` class

Discussion: Benefits of Exceptions

- Been talking about details…
- Why does Java have exceptions as part of the language?

Exceptions Summary

- Exception handling should be *exceptional*
  - It is expensive
- Try to *prevent* Runtime Exceptions
- Throw exceptions in your code for improved error handling/robustness
- If your code calls a method that throws a checked exception
  - Catch the exception if you can handle it well OR
  - Throw the exception to whoever called you and let them handle it

Assignment 5

- Practicing with Eclipse
- Inheritance, Collections
- Due Monday, October 30