Objectives

- Exceptions

Review

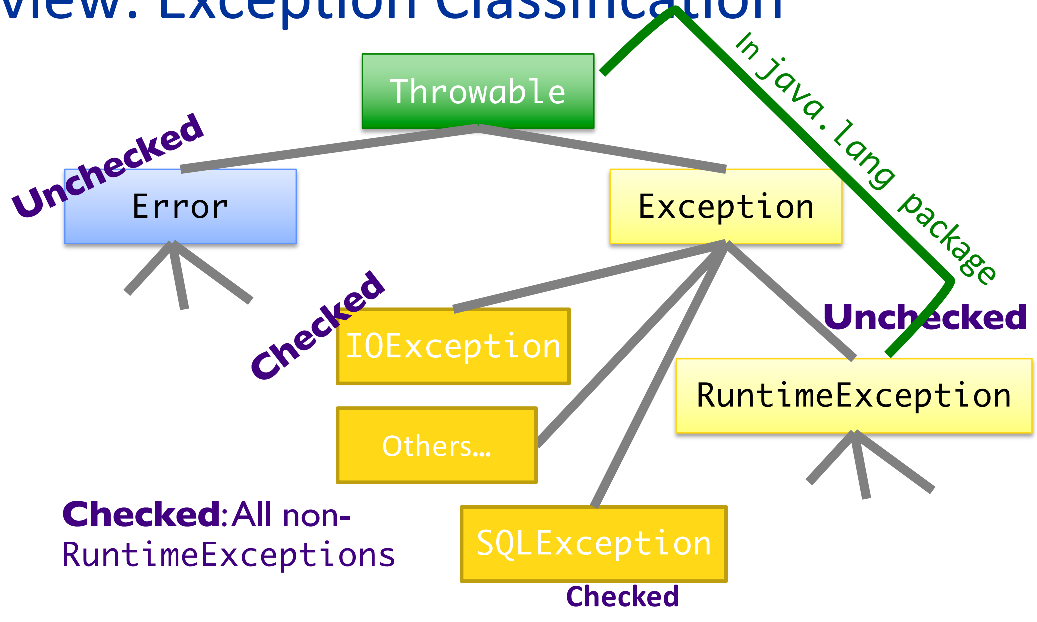
1. What are the benefits of the Collections Framework?
2. What is an Exception?
3. How do we create Exceptions?
4. How do we *advertise* that our method may produce an exception?
5. What are the different categories of exceptions?
   - What are examples (i.e., class names) of those categories of exceptions?
6. What is Eclipse? What can it do?
   - Why did I wait until now to show you Eclipse?

Review: Benefits of Collections Framework

- **Provides common, well-known interface**
  - Allows interoperability among unrelated APIs
  - Reduces effort to learn and to use new APIs for different implementations
- **Reduces programming effort:** provides useful, reusable data structures and algorithms
- **Increases program speed and quality:** provides high-performance, high-quality implementations of data structures and algorithms; interchangeable implementations → tuning
- **Reduces effort to design new APIs:** use standard collection interface for your collection
- **Fosters software reuse:** New data structures/algorithms that conform to the standard collection interfaces are reusable

3

Review: Exception Classification



Checked: All non-RuntimeExceptions

4

Review: 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)

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?

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

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` should **not** be documented as they are unpredictable

Eclipse Tradeoffs

- Very helpful – *after* you know what you're doing
  - You know
    - Code is compiled before executed
    - Structure of classes
    - How to fix errors
- Eclipse can handle the "routine" for you
  - That wasn't "routine" for you a few weeks ago
  - Help you focus on the important design considerations
- Gives suggestions for fixes
  - **You need to think through what the appropriate fix is**
    - Sometimes, it's "I'm not done yet"
  - Don't say "Eclipse made me do <something>"
- Eclipse is a beast (memory hog)
  - If you have less than ~8GB of memory, Eclipse will be slow

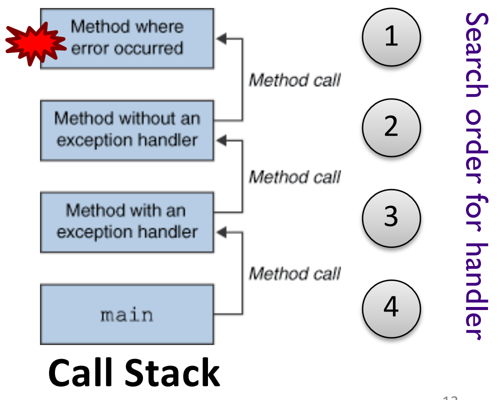
HANDLING EXCEPTIONS

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

13

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;
}
…
```

14

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 (ExceptionType e) {
        error code for
        ExceptionType
}
catch (ExceptionType2 e) {
        error code for
        ExceptionType2
}
```

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

- 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

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();
        }
}
```

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

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

FinallyTest.java

Practice: try/catch/finally Blocks

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

- 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;
}
```

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

21

Fun Fact: Python also has `finally`

```python
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

22

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

23

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;
}
```

24

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

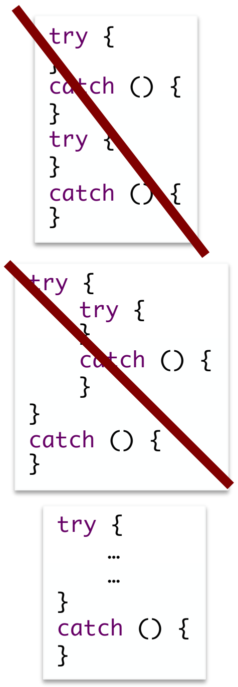
Summary: Methods Throwing Exceptions

- API documentation tells you if a method can throw an 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

30

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 () {
}
```

31

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

32

Discussion: Benefits of Exceptions

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

36

Benefits of Exceptions

- Force error checking/handling
  - Otherwise, won't compile
  - Does not guarantee "good" exception handling
- Ease debugging
  - Stack trace
- Separates error-handling code from "regular" code
  - Error code is in catch blocks at end
  - Descriptive messages with exceptions
- Propagate methods up call stack
  - Let whoever "cares" about error handle it
- Group and differentiate error types

Exceptions Summary

- Try to prevent Runtime Exceptions
- Throw Exceptions in your code for improved error handling/robustness
- If your code calls a method that throws an exception
  - Catch the exception if you can handle it well OR
  - Throw the exception to whoever called you and let them handle it

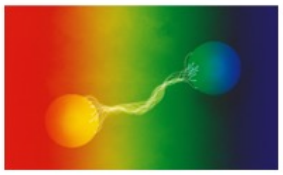
Extra Credit Opportunity

Office of the Dean presents 2022 Nobel Symposium

The Nobel Prize in Physics :

Quantum Information Science

Speaker: Tom Marcais



"for experiments with entangled photons, establishing the violation of Bell inequalities and pioneering quantum information science."

Wednesday, **October 26, 12:15-1:15**
Harte Center for Teaching and Learning (Leyburn 128)

Refreshments provided

Oct 21, 2022

Post summary on Canvas discussion forum

39

Assignment 5

- Practicing with Eclipse
- Inheritance, Collections
- Due next Friday

Oct 21, 2022 Sprenkle - CSCI209 40

40