EXCEPTIONS

EXCEPTION VS ERROR

- **exception** = object that defines unusual or erroneous situation
    - "thrown" by a program / runtime environment
    - often "caught" and handled in other parts of the code
- **error** = generally represents uncoverable situation

EXCEPTION EXAMPLES

- Divide by 0 (`ArithmeticException`)
- Using an index out of the bounds of an array (`ArrayIndexOutOfBoundsException`)
- Can't find specified file (`FileNotFoundException`)
- Other I/O issues (`IOException`)
- Try to dereference a null (`NullPointerException`)
- Try to access character beyond length of string (`StringIndexOutOfBoundsException`)

ERRORS

- Generally pretty crucial where only reasonable approach is to terminate problem
- Examples:
  - Out of memory (`java.lang.OutOfMemoryError`)
  - Stack overflow (`java.lang.StackOverflowError`)

PROCESSING EXCEPTIONS

- do nothing
- handle it where it occurs
- handle it at a different point

UNCAUGHT EXCEPTIONS

- program will terminate abnormally
- produces message that describes what occurred and where
- shows "call stack trace" - basically traces back up the path of methods called that caused the exception to occur

TRY-CATCH STATEMENTS

```
try {
    //code that might throw exception
}
catch (IllegalArgumentException e) {
    //do something for this type of exception
}
catch (AnotherExceptionType e) {
    //do something for this type of exception
}
finally {
    //code to happen regardless of whether there was exception
}
```

TRY-CATCH STATEMENTS

- at most one `catch` clause will be triggered
- always triggers the **first** appropriate where the exception `is instanceof` the specified exception type
- careful with inheritance

TRY-CATCH

- `finally` is optional
- can have one or more `catch`
- upon exception, control transfers to first `catch` that corresponds class of exception thrown

EXCEPTION PROPAGATION

- don't necessarily have to catch and handle exception where it occurred
- not caught - control returns to calling method
    - not caught in calling, control returns to method that called method that called...
- exceptions are propagated until caught/handled or passed out of main (uncaught exception)
- catch at higher level by enclosing method invocation with `try-catch`

CHECKED VS UNCHECKED

- unchecked = generated at runtime
  - not possible for compiler to tell code will handle exception
- checked
  - compiler confirms that method might throw exception
  - compiler requires code that calls method to use `try-catch`
  - alternative, method must specify it throws with `throws`

WHAT ARE EXCEPTIONS?

- Like anything else in Java -- classes
- Hierarchy of exceptions related by inheritances
- Like everything else, explicitly extends from Object
    - `Throwable` = parent of `Error` and `Exception`
    - Additional different levels