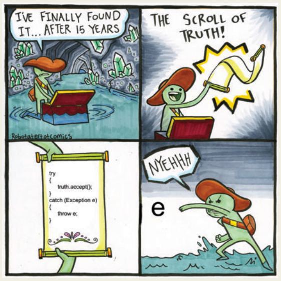


```
try
{
    truth.accept();
}
catch (Exception e)
{
    throw e;
}
```

Meme Credit: u/ProgrammerPete, reddit

CS 2112

Recitation 3
Exceptions

September 10 / 11, 2024

Agenda

- Error Handling
- Throwables
- Exercise

Reminders

- A2 Out Now
- Design Doc
  Due Wednesday

ERROR HANDLING

```
try {
```

```
function MoveDown() {
	if (document.activeElement.getAttribute("data-isrep") == "true") {
		//If this is a response

		var El = document.activeElement;
		//NewInput.id = CurrRow + "ResponseInput" + CurrCol;
		//ResponseContent for Divs
		var CurrRow = Number(El.id.split("ResponseInput")[0]);
		var CurrCol = Number(El.id.split("ResponseInput")[1]);

		if (El.getAttribute("data-reprow") != null) {
			//If this is one of many responses to the same point
			var CurrRepNum = Number(El.getAttribute("data-reprow"));
			if (document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") + "-" + (CurrRepNum + 1)) != null) {
				document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") + "-" + (CurrRepNum + 1)).focus();
				return;
			} else {
				//If this is the last response of many responses
				CurrCol = Number(El.parentElement.id.split("ResponseContent")[1]);
			}

		} else if (document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") + "-1") != null) {
			document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") + "-1").focus();
			return;
		}

		/*if(document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") +  (CurrRepNum + 1)) != null){
		document.getElementById(El.parentElement.id.replace("ResponseContent", "ResponseInput") + (CurrRepNum + 1)).focus();
	} else*/
		if (document.getElementById((CurrRow + 1) + "ResponseInput" + CurrCol) != null) {
			document.getElementById((CurrRow + 1) + "ResponseInput" + CurrCol).focus();
		} else if (document.getElementById("Input" + (CurrRow + 1)) != null) {
			document.getElementById("Input" + (CurrRow + 1)).focus();
		}

		return;
	}
	var ElNum = GetElNum();
	if (document.getElementById("Input" + (ElNum + 1)) != null) {
		document.getElementById("Input" + (ElNum + 1)).focus();
	}
}
```

```
} catch (e) {
  // Silently ignore error and hope it goes away
}
```

It's tempting to view Exceptions as the problem that needs to be fixed. You may even be tempted to wrap everything in a try catch to make the Exceptions stop. But Exceptions are just the symptom of an underlying issue – that your code is broken. And they're actually super userful.

Motivating Example

```
List o = { "Latte", "Espresso" };

    int x = o.indexOf("Lattee")

            x ⇒ -1
```

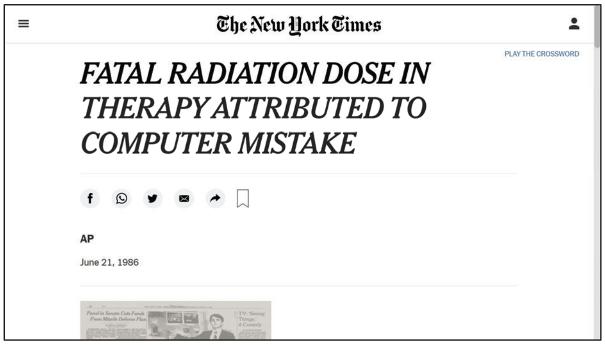
If I spell the item I'm looking for wrong, indexOf returns -1

```
// many lines of code later


while (x != 0) {

    dispenseCoffee();

    x--;

}
```

This code is going to call dispenseCoffee() a few billion times. Your coffee cup is going to overflow.

```
// many lines of code later


while (x != 0) {

dispenseRadiation();

    x--;

}
```

This is suddenly much less funny



The New York Times

PLAY THE CROSSWORD

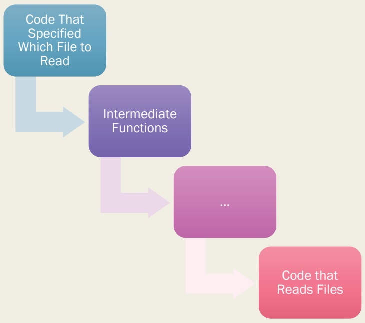
FATAL RADIATION DOSE IN THERAPY ATTRIBUTED TO COMPUTER MISTAKE

AP

June 21, 1986

A crash is NOT the worst thing that can happen

Continuing execution in an invalid state can be catastrophic



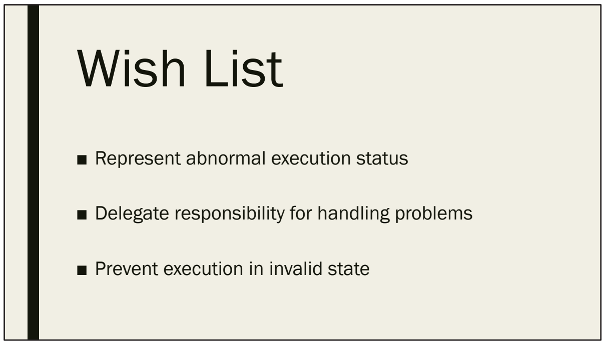
Code That Specified Which File to Read

Intermediate Functions

...

Code that Reads Files

Making things harder is that often times, the code that encounters the error (eg reading a nonexistent file) and the code that is responsible for the error (eg specifying which file to read) don't live in the same place and may not have even been written by the same person or team

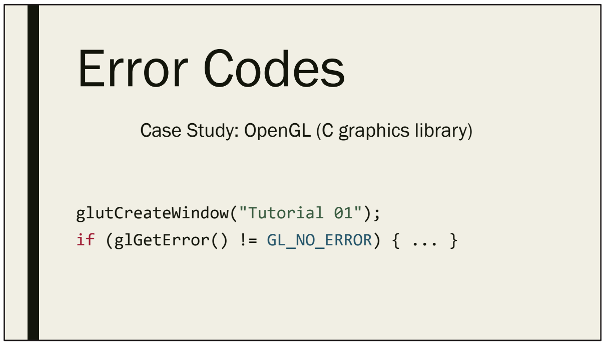


Wish List

- Represent abnormal execution status
- Delegate responsibility for handling problems
- Prevent execution in invalid state

As such, here are some features we'd like in our way of modelling problems

ATTEMPT 1:
ERROR CODES

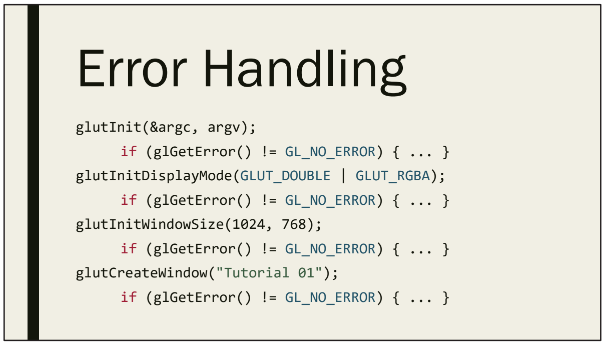


Error Codes

Case Study: OpenGL (C graphics library)

```
glutCreateWindow("Tutorial 01");
if (glGetError() != GL_NO_ERROR) { ... }
```

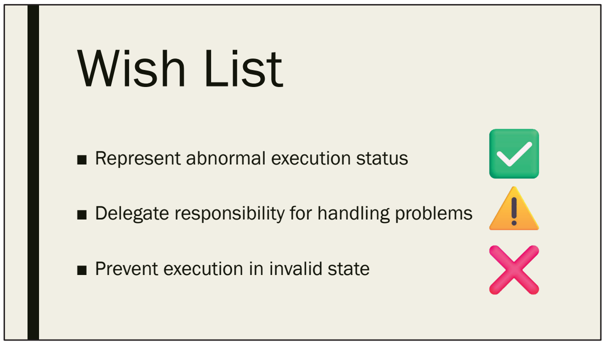
Some old, low level libraries will provide either a function or a return value that is some number if successful and a different number for errors



Error Handling

```
glutInit(&argc, argv);
     if (glGetError() != GL_NO_ERROR) { ... }
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
     if (glGetError() != GL_NO_ERROR) { ... }
glutInitWindowSize(1024, 768);
     if (glGetError() != GL_NO_ERROR) { ... }
glutCreateWindow("Tutorial 01");
     if (glGetError() != GL_NO_ERROR) { ... }
```

You end up checking for these error codes all over the place



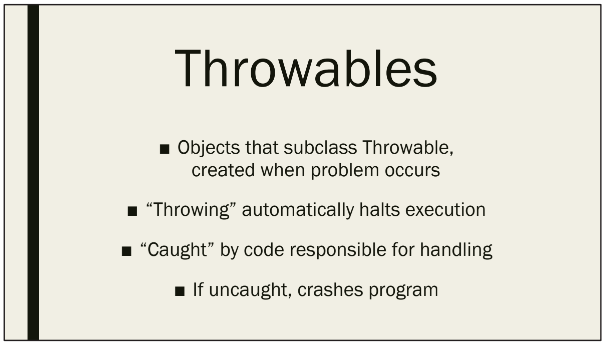
Wish List

- Represent abnormal execution status ✓
- Delegate responsibility for handling problems ⚠
- Prevent execution in invalid state ✗

We do have a way of representing problems. However, if the place to handle these problems lives further away from the callsite, it's still not easy to delegate. And there's nothing that this does if the programmer forgets to check for the status code.

ATTEMPT 2: THROWABLES

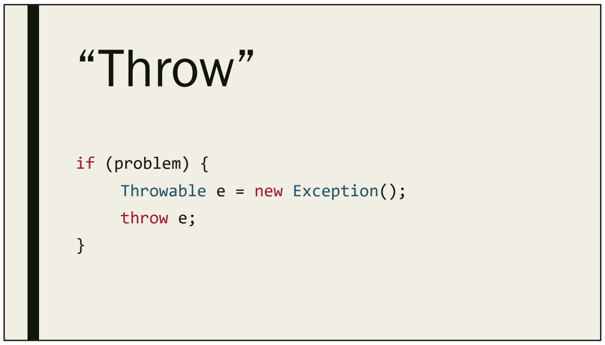
So instead this is how Java does it



Throwables

- Objects that subclass Throwable, created when problem occurs
- "Throwing" automatically halts execution
- "Caught" by code responsible for handling
- If uncaught, crashes program

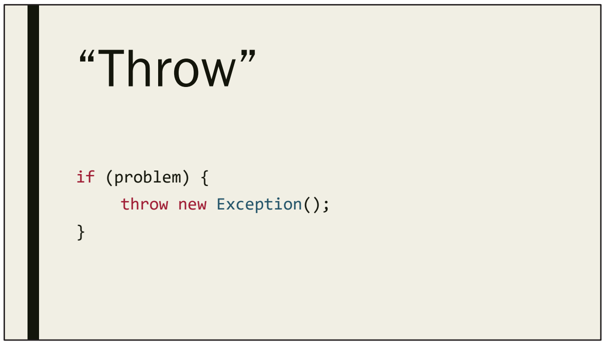
An object of type Throwable is a special object that can be "thrown" to halt execution due to an issue



"Throw"

```
if (problem) {
    Throwable e = new Exception();
    throw e;
}
```

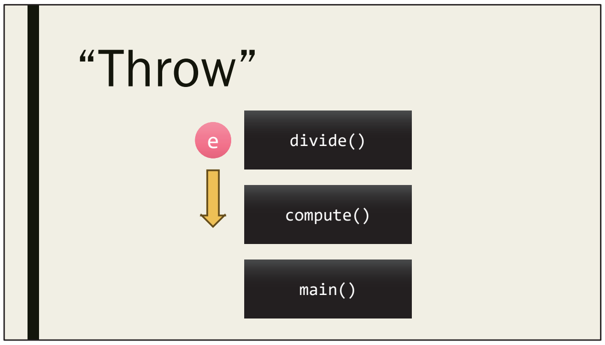
When a problem occurs that you'd like to pass off, create a new object of type Throwable like any other, and then pass that object to the "throw" keyword.



“Throw”

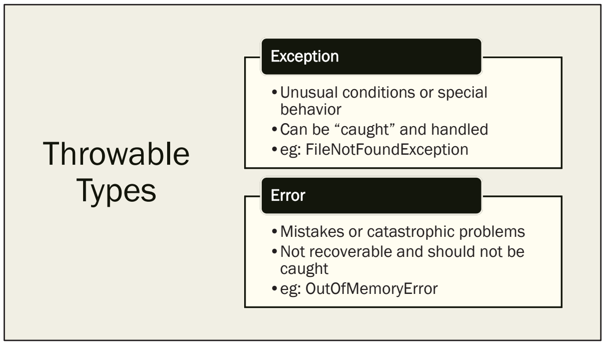
```
if (problem) {
    throw new Exception();
}
```

Often, we skip the step of assigning it to a variable first. However, don’t forget the “new” as we are instantiating a new object.



“Throw”

The throwable is then “thrown” down the callstack, waiting to be caught by the first method that tries to “catch” it. If it’s not caught, the program crashes.



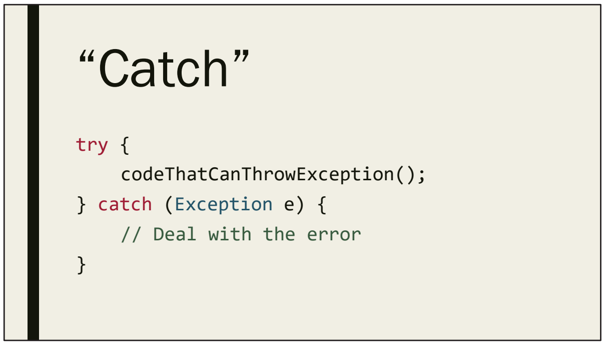
Throwable Types

Exception

- Unusual conditions or special behavior
- Can be “caught” and handled
- eg: FileNotFoundException

Error

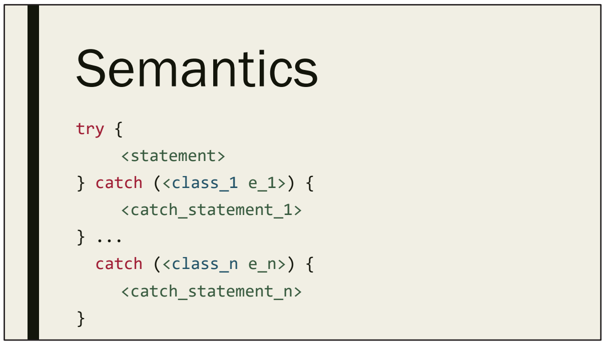
- Mistakes or catastrophic problems
- Not recoverable and should not be caught
- eg: OutOfMemoryError



“Catch”

```
try {
    codeThatCanThrowException();
} catch (Exception e) {
    // Deal with the error
}
```

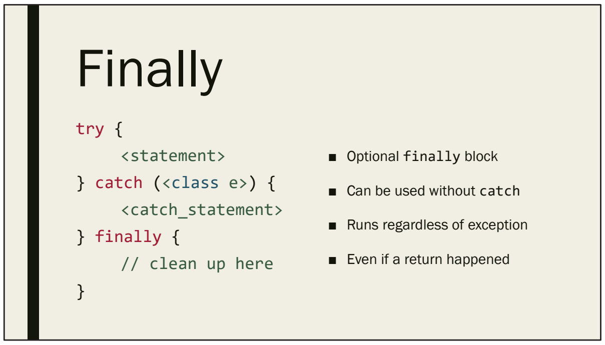
When you want to take responsibility for and handle the problems that may occur in another block of code, wrap that code in a “try” and you can catch the exception with a catch block.



Semantics

```
try {
    <statement>
} catch (<class_1 e_1>) {
    <catch_statement_1>
} ...
  catch (<class_n e_n>) {
    <catch_statement_n>
}
```

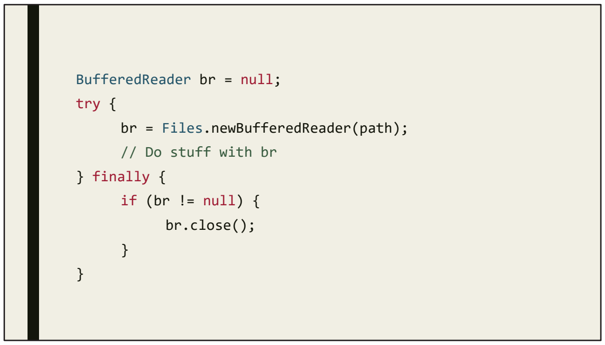
If an exception is thrown, the first catch block whose declared type is a supertype of the thrown exception will run. Multiple catch blocks can be chained.



Finally

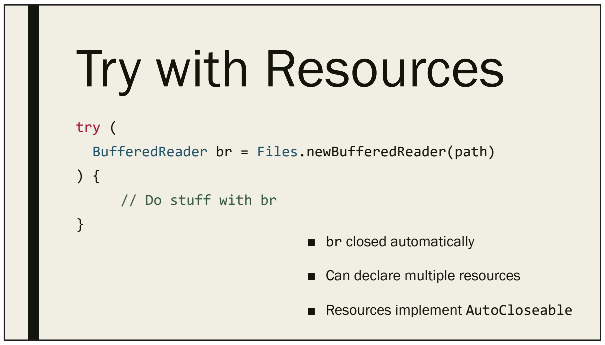
```
try {
    <statement>
} catch (<class e>) {
    <catch_statement>
} finally {
    // clean up here
}
```

- Optional `finally` block
- Can be used without `catch`
- Runs regardless of exception
- Even if a return happened



```
BufferedReader br = null;
try {
    br = Files.newBufferedReader(path);
    // Do stuff with br
} finally {
    if (br != null) {
        br.close();
    }
}
```

Finally blocks are often used to clean up resources when done

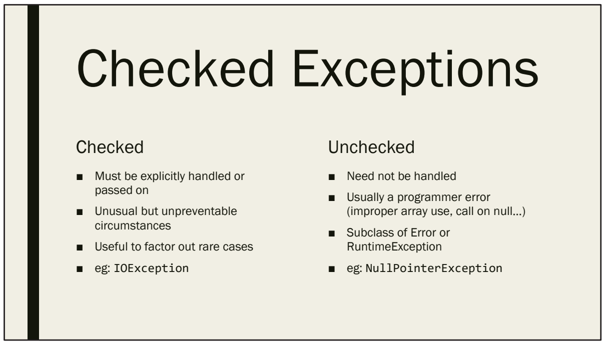


Try with Resources

```
try (
  BufferedReader br = Files.newBufferedReader(path)
) {
    // Do stuff with br
}
```

- br closed automatically
- Can declare multiple resources
- Resources implement `AutoCloseable`

This is newer syntax that does the same thing as the previous slide.
The new syntax is preferable especially if you have multiple resources, as if the close() method on one of them throws an exception, this will still make sure the others are closed properly.



Checked Exceptions

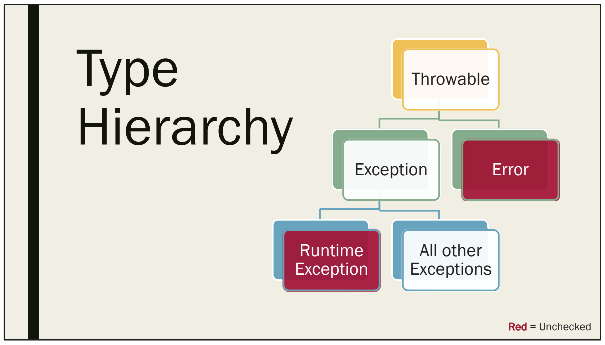
Checked

- Must be explicitly handled or passed on
- Unusual but unpreventable circumstances
- Useful to factor out rare cases
- eg: `IOException`

Unchecked

- Need not be handled
- Usually a programmer error (improper array use, call on null…)
- Subclass of Error or RuntimeException
- eg: `NullPointerException`

Exceptions come in two types



Type Hierarchy

Throwable

Exception

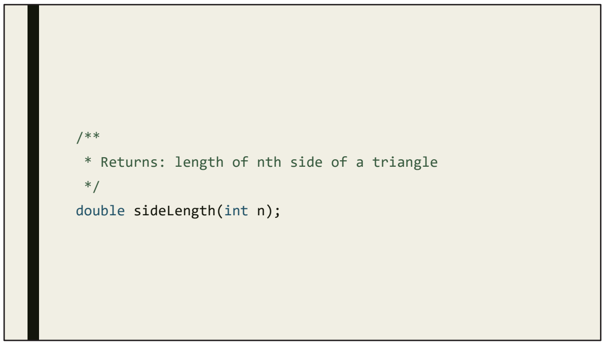
Error

Runtime Exception

All other Exceptions

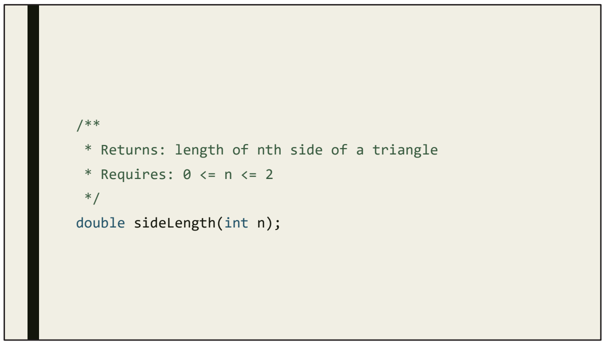
Red = Unchecked

EXAMPLES



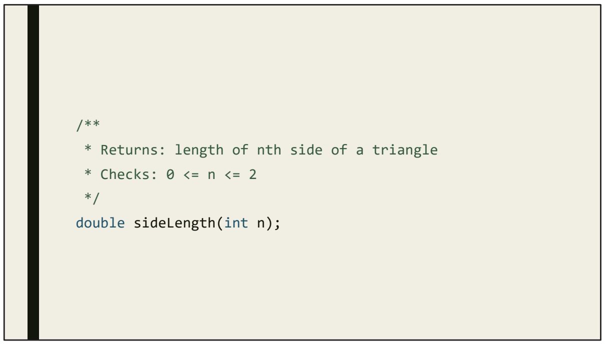
```
/**
 * Returns: length of nth side of a triangle
 */
double sideLength(int n);
```

What's wrong with this?

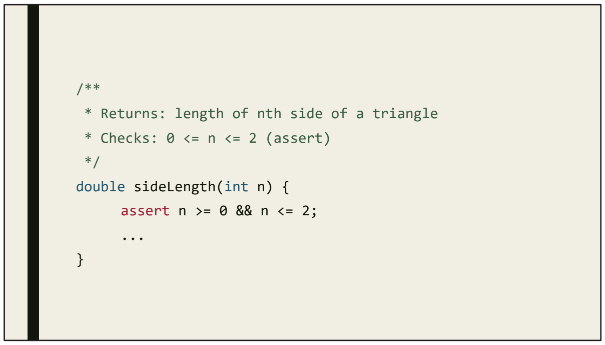


```java
/**
 * Returns: length of nth side of a triangle
 * Requires: 0 <= n <= 2
 */
double sideLength(int n);
```

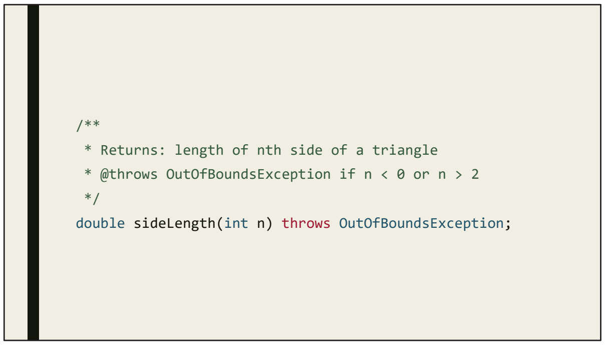
The following variants were discussed in class



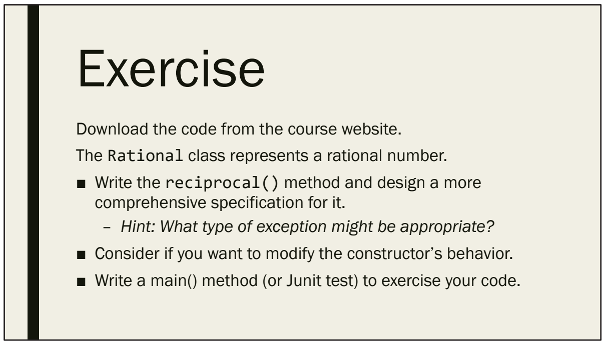
```java
/**
 * Returns: length of nth side of a triangle
 * Checks: 0 <= n <= 2
 */
double sideLength(int n);
```



```java
/**
 * Returns: length of nth side of a triangle
 * Checks: 0 <= n <= 2 (assert)
 */
double sideLength(int n) {
      assert n >= 0 && n <= 2;
      ...
}
```



```java
/**
 * Returns: length of nth side of a triangle
 * @throws OutOfBoundsException if n < 0 or n > 2
 */
double sideLength(int n) throws OutOfBoundsException;
```



Exercise

Download the code from the course website.

The `Rational` class represents a rational number.

- Write the `reciprocal()` method and design a more comprehensive specification for it.
  - *Hint: What type of exception might be appropriate?*
- Consider if you want to modify the constructor's behavior.
- Write a main() method (or Junit test) to exercise your code.

CS 2112