Exceptions

CSE 331: Software Design and Implementation, Autumn 2026

Your job is to deliver code that you know works.

TODO

- Reading 3: due Friday, October 30, 11:59pm
- Homework 3: due Friday, November 6, 11:59pm
- Section Tuesday: exceptions and subtyping

Two questions for the hour

- **Control** is where the program is: the line running now
- At any line: **where does control go from here?**
- At any line: **how could control have reached this point?**
- A claim at a point holds only if it is true each time a run reaches it
  - *However* the run got there

Half sheet, question 1 (3 minutes)

- `Trace.java` is on your program sheet
- Write what it prints, one line of output per line
- Two minutes alone, then one with a neighbor
- Step through the run

The trace is the stack

```
$ java Trace
⋮
Exception in thread "main" java.lang.ArithmeticException:
    cannot divide by zero
 at Trace.g(Trace.java:27)
 at Trace.f(Trace.java:21)
 at Trace.main(Trace.java:15)
```

- `g` at line 27: the throw itself
- `f` at line 21: where it had called `g`
- `main` at line 15: where it had called `f`

Where does control go from here?

- A **frame**: what Java keeps for one call while it runs
  - The method's variables, and the line it is at
- The **call stack**: the frames of the calls not yet returned, newest on top
  - A call pushes a frame; a return pops the top one
- Stand on line 21, `g(n);`: where does control go next?
  - Line 22 if `g` returns; out of `f` if `g` throws
- **Every call is a place control can come back to, and leave through**

Half sheet, question 2 (3 minutes)

- `TraceCatch.java`'s `main` is on your program sheet; `f` and `g` are Trace's
- Does `f: done` print for the call `f(2)`? For `f(0)`?
- Does `main: done` print?
- Step through the run

Rule: how a throw finds its catch

- A `throw` looks for a handler frame by frame down the stack
- Its own frame first; then, at the line each frame is at:
  - Inside a `try` with a `catch` for the exception's class? Control goes to it
  - And the method goes on after the `try`
  - Otherwise the method ends, and the frame below is next
- No frame left: Java prints the stack trace, and the program stops

A block that runs either way

```
18  static void f(int n) {
19      System.out.println("  f: calling g(" + n + ")");
20      try {
21          g(n);
22      } finally {
23          System.out.println("  f: finally");
24      }
25      System.out.println("  f: done");
26  }
```

- `main` keeps its `catch`; `f`'s `try` has only a `finally`: what prints?
- A `finally` runs on the way out, whichever way out it is

The calculator, done one way

```
57   try {
58       step(stack, token);
59   } catch (CalcStack.CalcException e) {
60       System.out.println("error: " + e.getMessage());
61   }
```

- Lecture 6's calculator said no six ways; this one says no one way
- An interface, an implementation, the app: the shape of HW3's starter
- `step` handles nothing; this one `catch` takes every throw in the stack

A catch for Exception: predict what prints

```
 5  public class TraceSwallow {
 6      public static void main(String[] args) {
 7          try {
 8              System.out.println("main: calling f(2)");
 9              f(2);
10              System.out.println("main: calling f(0)");
11              f(0);
12          } catch (Exception e) {
13          }
14          System.out.println("main: done");
15      }
```

- A `catch` for Exception takes a bug's throw as much as a user's

A method with two exits

```
60  /**
61   * @modifies this
62   * @effects pops the top two numbers (first and second) and pushes
63   *     second / first, rounded toward zero
64   * @throws TooFewOperandsException if the stack has fewer than two
65   *     numbers, and leaves the stack unchanged
66   * @throws DivideByZeroException if the top number is zero, and
67   *     leaves the stack unchanged
68   */
69  void divide();
```

- Each `@throws` says what its exit leaves: **the stack unchanged**
- The test: `assertThrows`, then `assertEquals` on what is left

Half sheet, question 3 (3 minutes)

- The calculator prints the stack on `p`
- A user types `6 0 /`, sees `cannot divide by zero`, then types `p`
- What should it print?
- With `DraftStack`'s `divide` (program sheet), what does it print?

The claim at the catch

```
{{ P }}
try {
    body
    {{ Q }}
} catch (X e) {
    {{ E }}
    handler
    {{ Q }}
}
{{ Q }}
```

- `E` must be true at every throw of an `X` that can reach this `catch`
- At `main`'s `catch`: *the stack is as it was before this command*

Two questions, answered

- **Where does control go from here?**
  - To the next line, or out of the method, through any call
- **How could control have reached this point?**
  - At a `catch`: from every throw under its `try`
- The claim at the point has to be true however control got there
  - Make it true at each throw, in the method that throws
- HW3: test what the inventory holds after the throw

Coming up

- **Friday**: subtyping
- **Section Tuesday**: exceptions and subtyping
- Reading 3: due Friday, October 30, 11:59pm
- Homework 3: due Friday, November 6, 11:59pm
- Hand in your half sheet at the door