CS1110 07 April 2008 Exceptions in Java.

Today's reading: Ch. 10. Next lecture's reading: Ch 17.

(Most) on-duty consultants now identifiable by stylish headgear.

Prelim 3 next *Tuesday* (April 14), 7:30-9pm, *Ives 305*

- See "about Prelim III" handout
- Pick up graded quizzes from last time (up front)
- There is a review session Sunday April 12th, 1-3pm, Phillips 101. Slides will be posted on the website.
- Prelim 3 is, like Prelim 2, cumulative:
  - Look over your Prelim 2 and the P2 solutions (posted to the website, with fix of edit error in solution to last question).
    - Uncollected P2s can be retrieved from Upson 360, M-F 10am-noon and 2-4pm with ID card.

A6 due Saturday.

FYI: we promised to tell you: the first weighted-die code we saw (March 10) maintains the invariant "Either r >= iStart or r is in segment i-1, where iStart is the start of segment i".

1

Today's topic: when things go wrong (in Java)

Q: What happens when an error cause the system to abort?

(NullPointerException, ArrayIndexOutOfBoundsException, …)

Understanding this helps you debug.

Q: What if termination isn't the right thing to do?

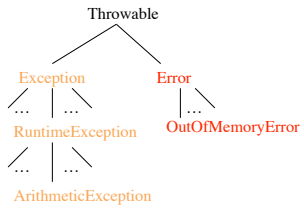
Understanding this helps you write more flexible code.

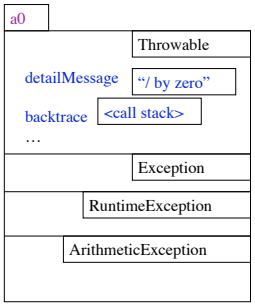
Important example: a "regular person" enters malformed input.

It is better to warn and re-prompt the user than to have the program crash (even if they didn't follow your exquisitely clear directions).

2

errors (little e) cause Java to *throw* a Throwable object





Exceptions are signals that help may be needed; they can be "handled".

Errors are signals that things are beyond help.

3

The Throwable is thrown to successive "callers" until *caught*. Here, Java catches it because nothing else does.

Ex.first();

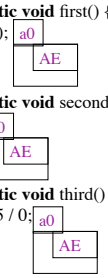
System prints the call-stack trace on catching exception:

```
ArithmeticException: / by zero
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
  at sun.reflect.NativeMethodAccessorImpl.
          invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(…)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(…)
  at java.lang.reflect.Method.invoke(Method.java:585)
```

```
/** Illustrate exception handling */
public class Ex {
    public static void first() {
        second();
    }

    public static void second() {
        third();
    }

    public static void third() {
        int x= 5 / 0;
    }
```



4

Try-statements catch and handle Throwables.

```
/** = recip(x) + recip(x*x), or -1 if x is 0*/
public static double computeResult(int x) {
  try {
      return recip(x) + recip(x*x);
  } catch (ArithmeticException ae) {
      return -1;
    }
}

/** = reciprocal of x.  Thows an ArithmeticException if x is 0.
 (suppose this is third-party code that you can't change)*/
public static double recip(int x) {
    …;
}
```

Execute the try-block. If it finishes without throwing anything, fine.

If it throws an ArithmeticException object, catch it (execute the catch block); else throw it out further.

5

Try-statements vs. if-then checking

```
/** = recip(x) + recip(x*x), or -1 if  x is 0*/
public static double computeResult(int x) {
  If (x != 0) {
      return recip(x) + recip(x*x);
  } else {
      return -1;
    }
}
```

This was meant to be a small example. Use your judgment:

- For (a small number of) simple tests and "normal" situations, if-thens are better.
- If the caller, not the method itself, should decide what should be done, throw an exception (like recip() does).
- There are some natural try(catch) idioms…

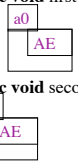
6

We can create new Throwable objects ourselves.

```
/** Illustrate exception handling */
public class Ex {
    public static void first() {
      second();
    }

    public static void second() {
      third();
    }

    public static void third() {
      throw new
        ArithmeticException
                ("I threw it");
    }
}
```




```
Ex.first();
ArithmeticException: I threw it
  at Ex.third(Ex.java:14)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(…)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(…)
  at java.lang.reflect.Method.invoke(Method.java:585)
```

7

We can write our own Exception subclasses, but we may need a "throws" clause to compile

```
/** Class to illustrate exception handling */
public class Ex {
   public static void first() throws MyException {
     second();
   }
   public static void second() throws MyException {
     third();
   }
   public static void third() throws MyException {
     throw new MyException("mine");
   }
```

Don't worry about whether to put a throws clause in or not. Just put it in when it is needed in order for the program to compile.

8