

CS1110 19 November 2009 Exceptions in Java.

Today's reading: Ch. 10. Next lecture's reading: sec. 9.3

A7 due Friday December 4.

Please check that your grades on CMS match what you think they are.

No labs Tuesday Nov 24 or Wed Nov 25; no office hours during Thanksgiving break. There *is* class Tuesday Nov 24.

The final exam will be Monday, Dec. 14th, 7-9:30pm in Baker Laboratory 200. We are scheduling review sessions for study week, Dec 7-9.

1

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

Q1: What happens when an error causes the system to abort?

(NullPointerException, ArrayIndexOutOfBoundsException, ...)

Understanding this helps you debug.

Q2: Can we make something other than termination happen?

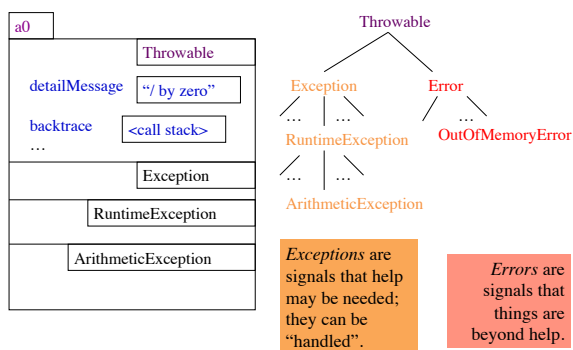
Understanding this helps you write more flexible code.

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

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

2

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



3

`Ex.first();`

*/** Illustrate exception handling */*

```
public class Ex {
    public static void first() {
        second();
    }
}
```

Throwable object is thrown to successive "callers" until caught. (Here, Java will catch it because nothing else does.)

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

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)
```

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

`a0` `AE`

4

How can we catch/handle Throwables? With *Try/catch* blocks.

/** = reciprocal of x. **Throws an ArithmeticException if x is 0.**
(Assume this is third-party code that you can't change.)*/

```
public static double reciprocal(int x) {
    ...;
}
```

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

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

/** = reciprocal(x), or -1 if x is 0*/

```
public static double ourReciprocal2(int x) {
    if (x != 0) {
        return reciprocal(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 that the method itself should handle, if-thens are better.
- If the caller, not the method itself, should decide what should be done, throw an exception (like `reciprocal()` does) to signal the caller.
- There are some natural try/catch idioms...

6

We can create new objects of pre-existing Throwable subclasses.

/** Illustrate exception handling */

```
public class Ex {
    public static void first(int x) {
        second(x+1);
    }

    public static void second(int y) {
        third(y+1);
    }

    public static void third(int z) {
        throw new
            ArithmeticException
                ("third: z was " + z);
    }
}
```

Ex.first(3);

ArithmeticException: third: z was 5
 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 even 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 OurException {
        second();
    }

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

    public static void third() throws OurException {
        throw new OurException("intentional error at third");
    }
}
```

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