Announcements for This Lecture

Prelim

- Generally grades are good
  - Mean: 80, Median 84
  - Similar to last semester
  - 62 is (probably) C- and below
- In Upson 360 by Thursday
  - Check that grade is in CMS!

Assignments

- A4 due Thursday
  - Do not wait until last minute
  - **Remember to report your time in the comments!**
  - Graded when you get back
- A5 posted Thursday
  - Have 1.5 weeks after Spring Break to do it
  - Welcome, but not expected, to do it over the break

Reading Today

- Chapter 10 (in entirety)

Exceptional Circumstances

```
/** Yields: the decimal number represented by s. */
int parseInt(String s) { ... }
```

- …but what if s is "bubble gum"?

```
/** Yields: the decimal number represented by s, or -1
 *  if s does not contain a decimal number. */
```

- …but what if s is "–1"?

```
/** Yields: the decimal number represented by s
 *  Precondition: s contains a decimal number. */
```

- …but what if s might not, sometimes?
- Somehow, we have to be able to deal with the unexpected case

Dealing with Exceptional Circumstances

```
/** Yields: the decimal number
 *  represented by s.
 *  Pre: s contains a number. */
int parseInt(String s) { ... }

/** Yields: "s contains a number." */
boolean parseableAsInt(String s) { ... }
```

- Now we have to write:

```
if (parseableAsInt(someString)) {
   i = parseInt(someString);
} else {
   // do something about the error
}
```

- How to read a number from a file (in 14 easy steps):
1. Open the file
2. If the file doesn't exist, …
3. If there was a disk error, …
4. Read a line from the file.
5. If the file was empty, …
6. If there was a disk error, …
7. Convert string to a number.
8. If the string is not a number, …
9. If we have run out of memory, …
10. Close the file.
11. If there was a disk error, …
12. If t
13. If t
14. If t

Common Outcome

Weary programmers write code that ignores errors.

There has to be a better way!

Exception Handling

```
/** Parse s as a signed decimal integer.
 *  Yields: the integer parsed
 *  Throws: NumberFormatException is s not a number */
public static int parseInt(String s) ...
```

- What happens when parseInt finds an error?
  - Does not know what caused the error
  - Cannot do anything intelligent about it.
  - "throws the exception" to the calling method
  - The normal execution sequence stops!

Recovering from Exceptions

- try-catch blocks allow us to recover from errors
  - Do the code that is the try-block
  - Once an exception occurs, jump to the catch
- Example:

```
try {
   i = Integer.parseInt(someString);
   System.out.println("The number is: " + i);
} catch (NumberFormatException nfe) {
   System.out.println("Hey! That is not a number!")
}
```

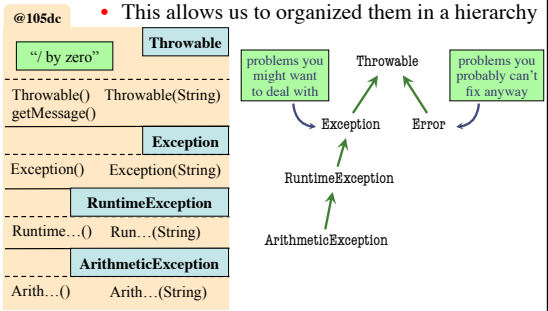
might throw a NumberFormatException

tells Java to handle N.F.E.s here

executes if the exception happens

Exceptions in Java

- Exceptions are instances of class Throwable
- This allows us to organized them in a hierarchy



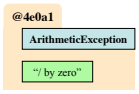
Exceptions and the Call Stack

- **Call:**

Ex.first();

- **Output:**

```
ArithmeticException: / by zero
 at Ex.third(Ex.java:13)
 at Ex.second(Ex.java:9)
 at Ex.first(Ex.java:5)
```




```
02 /** Illustrate exception handling */
03 public class Ex {
04    public static void first() {
05       second();
06    }
07
08    public static void second() {
09       third();
10    }
11
12    public static void third() {
13       int x = 5 / 0;
14    }
15 }
```

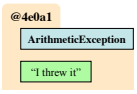
Exceptions and the Call Stack

- **Call:**

Ex.first();

- **Output:**

```
ArithmeticException: I threw it
 at Ex.third(Ex.java:13)
 at Ex.second(Ex.java:9)
 at Ex.first(Ex.java:5)
```




```
02 /** Illustrate exception handling */
03 public class Ex {
04    public static void first() {
05       second();
06    }
07
08    public static void second() {
09       third();
10    }
11
12    public static void third() {
13       throw new ArithmeticException ("I threw it");
14    }
15 }
```

Creating Your Own Exceptions

```
/** An instance is an exception */
public class OurException extends Exception {

   /** Constructor: an instance with message m*/
   public OurException(String m) {
      super(m);
   }

   /** Constructor: an instance with no message */
   public OurException() {
      super();
   }
}
```

This is all you need
- No extra fields
- No extra methods
- Just the constructors

throws and Checked Exceptions

- **Call:**

Ex.first();

- **Output:**

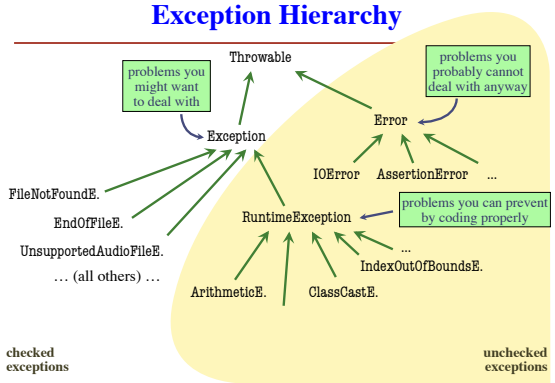
```
OurException: Whoa!
 at Ex.third(Ex.java:13)
 at Ex.second(Ex.java:9)
 at Ex.first(Ex.java:5)
```

throws clauses are required because OurException, unlike ArithmeticException, is a "**checked exception**."

Will not compile yet!

```
02 /** Illustrate exception handling */
03 public class Ex {
04    public static void first()  throws OurException {
05       second();
06    }
07
08    public static void second()  throws OurException {
09       third();
10    }
11
12    public static void third()  throws OurException {
13       throw new OurException("Whoa!");
14    }
15 }
```

Exception Hierarchy



throws and Checked Exceptions

```
public class Ex {
   public static void first() {
      try {
         second();
      } catch (OurException ae) {
         System.out.println("Caught it: " + ae);
      }
      System.out.println("Procedure first done.");
   }
   public static void second() throws OurException {
      third();
   }
   public static void third() throws OurException {
      throw new OurException("an error");
   }
}
```

- throws is needed if
  - The method itself throws checked exception
  - The method calls a method that throws a checked exception
- throws is **not** needed if
  - All checked exceptions are caught
  - Any uncaught exceptions are unchecked exceptions