Lecture 9

Exceptions

Types of Errors in Java

Syntactic Errors

- Can check at compile time
- Bad use of "grammar"

- **Examples:**
  - Lack of semicolon
  - Unknown method or variable
  - Use of method not in the apparent type of variable

Runtime errors

- Can only check at run time
- Generally have to do with contents (not type) of variable

- **Examples:**
  - Variable unexpectedly null
  - Bad downward casts
  - Method call that violates the parameter preconditions

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!")
}
```

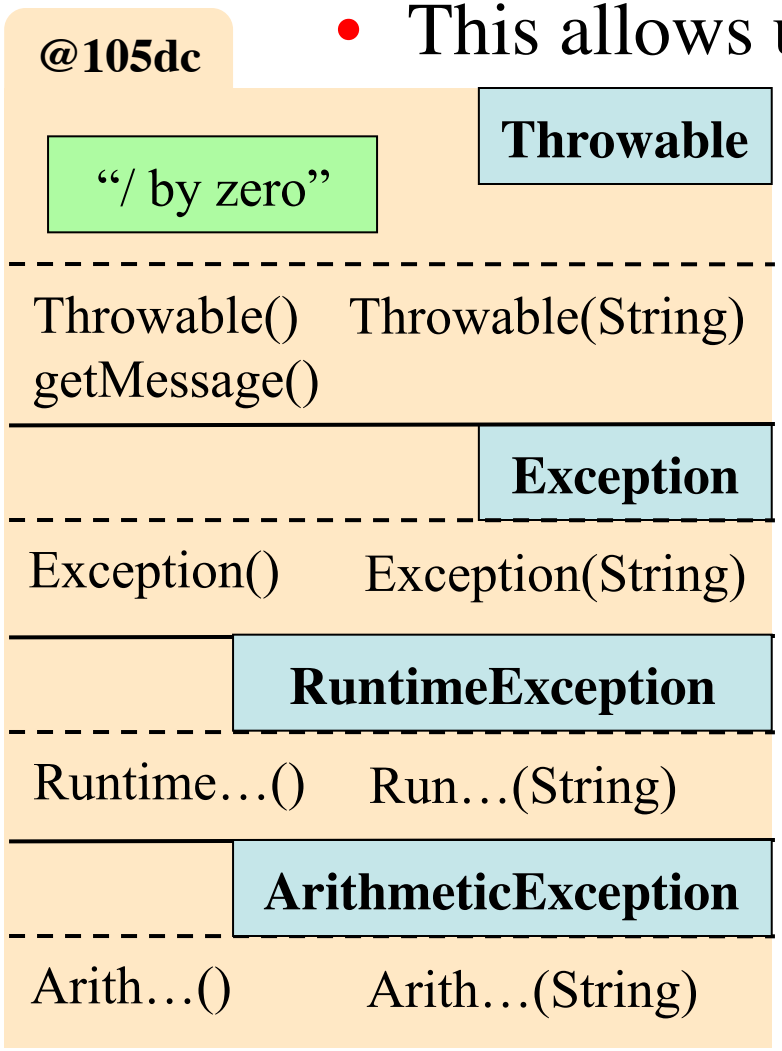
might throw a NumberFormatException

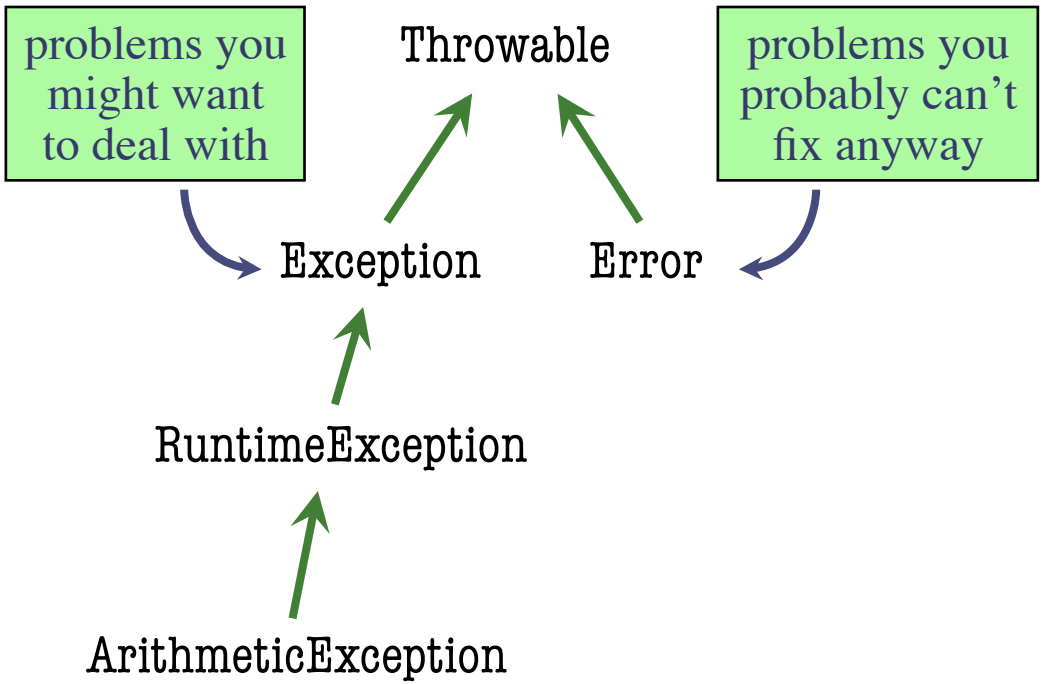
tells Java to handle N.F.E.s here

executes if the exception happens

Exceptions in Java

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





Creating Exceptions

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

}
```

Java creates Exception for you automatically

```
public static void foo() {
   throw new
        Exception("I threw it");
}
```

You create Exception manually by **throwing** it

Why So Many Exceptions?

```
public static int foo() {
    int x = 0;
    try {
      throw new RuntimeException();
      x  = 2;
    } catch (RuntimeException e) {
      x = 3;
    }
    return x;
}
```

- What is the value foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Why So Many Exceptions?

```
public static int foo() {
    int x = 0;
    try {
       throw new RuntimeException();
       x  = 2;
    } catch (Exception e) {
       x = 3;
    }
    return x;
}
```

- What is the value foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Why So Many Exceptions?

```
public static int foo() {
    int x = 0;
    try {
       throw new RuntimeException();
       x  = 2;
    } catch (ArithmeticException e) {
       x = 3;
    }
    return x;
}
```

- What is the value foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Java uses real type
to match Exceptions

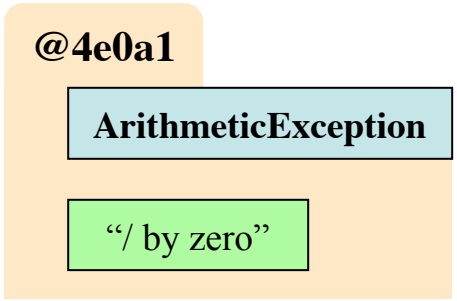
Exceptions and the Call Stack

- **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 }
```

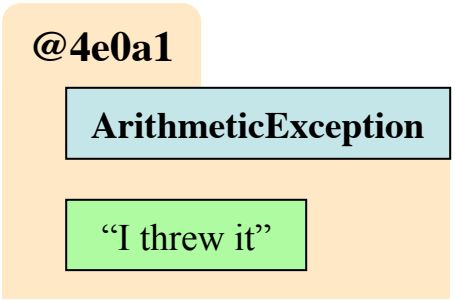
Exceptions and the Call Stack

- **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 }
```

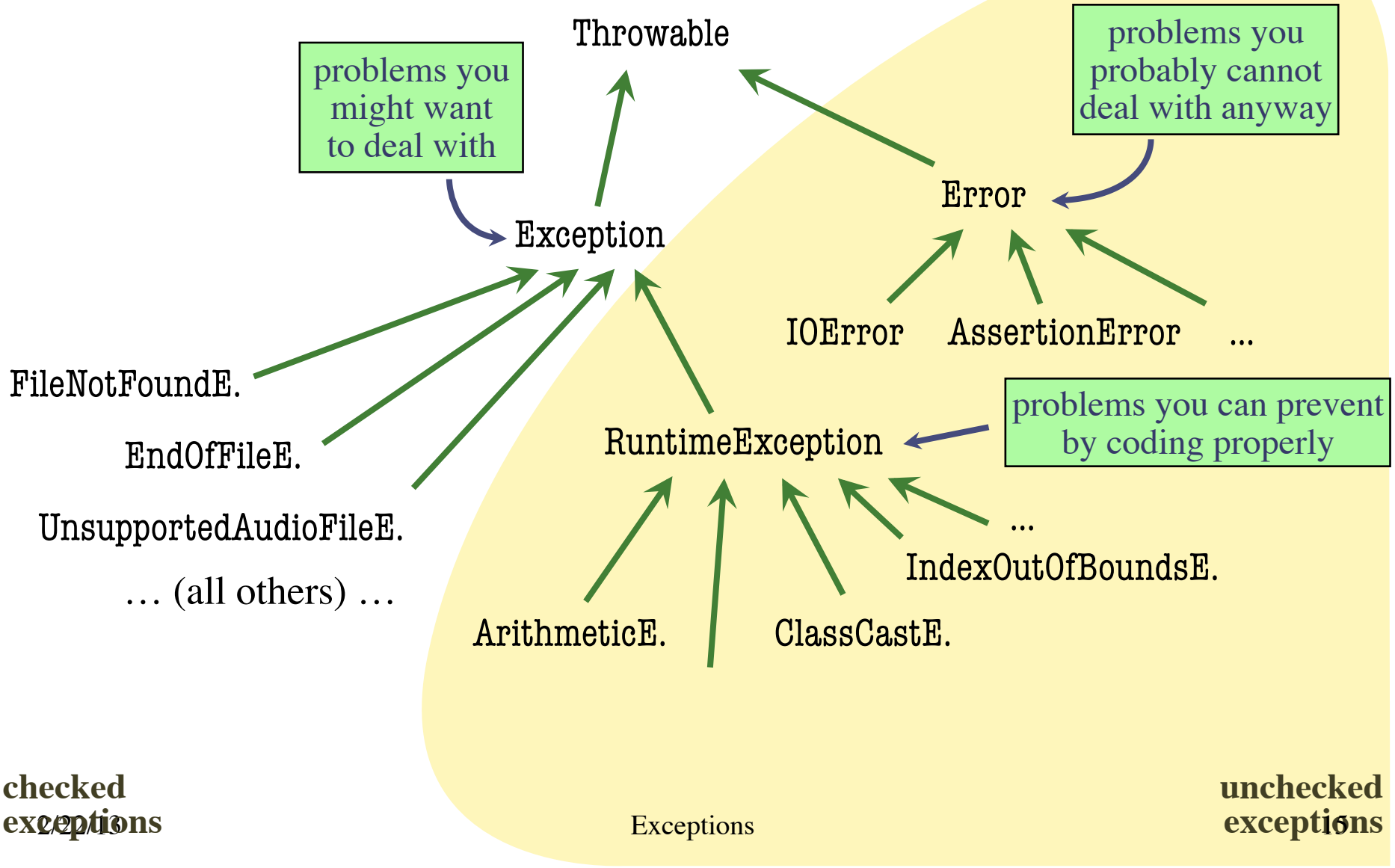
Creating Your Own Exceptions

```
/** 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

Exception Hierarchy



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**."

```
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 }
```

Will not compile yet!

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

Exceptions and the Java API

- Java API tells which methods throw exceptions
  - Look at the method description
  - Will list types of exceptions thrown and reason
- **Examples:**
  - `java.lang.String`
    - `charAt()` may throw `IndexOutOfBoundsException`
    - `endsWith()` may throw `NullPointerException`
  - `java.lang.Double`
    - `parseDouble()` may throw `NumberFormatException`
    - `compareTo()` may throw `ClassCastException`