Errors

the runtime system, which is responsible for finding code to handle the
an exception object, which describes the exception, and hands it off to
When such an error occurs within a Java method, the method creates

- Programming Error: Dereferencing null
- Hardware Error

Many kinds of errors can cause exceptions:

that disrupts the normal flow of instructions.

An exception is an event that occurs during the execution of a program.

Exceptions
What is an appropriate exception handler?

- Finds a method that contains an appropriate exception handler.
- Calls, beginning with the method in which the error occurred, until it
  calls, beginning with the method in which the error occurred, until it
  calls, beginning with the method in which the error occurred, until it
  calls, beginning with the method in which the error occurred, until it
  calls, beginning with the method in which the error occurred, until it

Where does the runtime system look?
- Groups error types and differentiates errors.
- Automatically propagates errors up the chain of method calls.
- Separates error handling code from regular code.

What are the advantages?

Exceptions (cont.)
... {
  "pizza\npizza\n(pizza\nprice,\nshape)\n: \nprice > \n0 \nor \nshape == \nnull"

  throw \nnew \nIllegalArgumentException()
}

if (\nprice > \n0 \n|| \nshape == \nnull)

  } \n
  (pizza\npizza\n(duplicate \nprice, \nashape \nshape)
...

  public \nclass \npizza

    Example

  Throwing Exceptions

January 26, 2001

Comp 212
These exceptions are handled or specified.

- **Error (not usually recovered from)**
  - RuntimeException (extends Exception)
- **Exception**
  - There are several predefined subclasses of Throwable.
  - Throwable contains a reference to a descriptive string.
  - The argument to throw must be a reference to an instance of a subclass

Thorwing Exceptions (cont.)
- ArrayIndexOutOfBoundsException
- NoSuchElementException
- NullPointerException
- OutOfMemoryException

The next two types are special. An instance of a subclass of these

Throwable Exceptions (cont.)
catching Exceptions

Example

```
shape): price > 0 or shape == null

        { System.err.println(e);
        }
        catch (java.util.ParseException e)
        { Not Reached.

        pizza = new Pizza(-4.69, null);
        }
```

January 26, 2001
Exceptions Can’t Be Ignored…

A method can’t ignore exceptions raised by another method that it calls. It must either...

- *catch* the exception or
- *specify* the exception.
```java
{ 
    call methodThatThrowsException;
    } method2 throws exception
    {
        {
            doErrorProcessing;
        } catch (exception) {
            call method2;
            try
            } method1
        }
```

Example

Exceptions Can’t Be Ignored... (cont.)
An arbitrary number of catch statements can follow the try statement.

```
{ ... }
{ catch (EXCTYPE e) { ... }
{ catch (EXCTYPE e) { ... }
{ catch (EXCTYPE e) { ...
neverCatchedMethod();
brokenMethodThrowsException(EXCTYPE2003());
} try
```

**Catching multiple exceptions**
throw multiple exceptions

A method can (potentially) throw an arbitrary number of exceptions (but not at once).
The `finally` statement enables a method to complete the method's execution. If there is cleanup code at the end of the method, it will never get called. Exceptions can cause control to leave the current method without designating code for execution even if an exception occurs.
The `finally` statement is not a handler. After `myCatcher()` is performed, the exception continues up the call chain in search of a handler.

Example:

```java
public class Example {

    public void method() throws Exception {
        try {
            throw new Exception();
        } catch (Exception e) {
            System.out.println("Caught exception: "+e.getMessage());
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```