



## Handling Exceptional Conditions

Slides by Wes Weimer

## Lecture Non-Goals

- Will **not** completely rehash CS 164 or Writing Solid Code
- Will **not** assume that you have never seen or handled an exception before
- Will **not** try to convince you that exceptions are great
- Will **not** bore you to tears with droning pontification ... we hope!
- So what will we talk about?

## Rumor has it ...

- ... that software engineering is important.
  - It's traditional to make a sacrificial offering
- 2000 US Job Count (NIST)
  - "computer programmer" 585,000
  - "software engineer" 697,000
- 2000 Median Salary (US Bur. Lab. Stat.)
  - "computer programmer" **\$57,590**
  - "software engineer: applications" **\$67,670**
  - "software engineer: systems" \$69,530

## All About Errors

- Software engineering has many aspects
- We'll skip requirements engineering, testing, design patterns, version control, runtime monitoring, UML, ...
- And just talk about:
  - run-time errors
  - exceptional situations
  - and how to handle them!

## Actual Lecture Goals

- Discuss "**exceptional situations**"
  - Importance, definitions, causes, high-level ideas ...
- Talk about "**error handling**"
  - Theory, practice, hints, ...
- Go over "**exception handling**"
  - Pros, cons, common pitfalls
- Work through "**error handling mistakes**"
  - Real, tricky, but avoidable
- Briefly show **some CS research**
- Give a bit of **further reading**
  - Links in PPT, slides available later
- Invite **class participation**

## Errors: Should We Care?

- 2002 US Real GDP:
  - \$10,080 billion
- 2000 US Software Sales:
  - \$180 billion
- 2002 US **Annual Cost of Software Errors**:
  - \$59.5 billion (**0.6% of GDP**)
- Answer: Yes
- (sources: NIST, US BLS, etc.)

## Error Handling: Should We Care?

- IBM Study: Up to two-thirds of a program can be devoted to error handling
- Our Survey: for Java, between 3% and 46% depending on program age and size
- Mistakes in which “**applications don't properly handle error conditions that occur during normal operation**” are reported as one of the top ten causes of Java web app security risks

## Bad News ...

- Exceptional situations are elusive!
- #1. Usually **difficult to foresee** them
  - ... or you would handle them and they wouldn't be problems
- #2. Boring or **hard to handle** them
  - PL Support, Software Engineering can help
- So what are these exceptions?

## Errors: What Are They?

- View from the OS / app barrier (x86/win32):
  - ✓ “Breakpoint”
  - ✓ “Access Violation”
  - ✓ “**Invalid Handle**”
  - ✓ “Illegal Instruction”
  - ✓ “Privileged Instruction”
  - ✓ “Stack Overflow”
  - ✓ “**Software Exception**” (VC++ try/catch/throw)

## Errors: What Are They?

- View from the app / user barrier:
- **InvalidObjectException**
  - “This exception occurs if you try to serialize a disconnected Hibernate Session and deserialize it in a different VM, or, if the classloader has restarted, for example, during hot redeployment in your application server or web container. This is especially visible in Tomcat.” (Hibernate, docnode 212)
- This is not good!

## Errors, Mistakes, and Handling

- **Run-Time Error**: Signals an exceptional or unexpected situation in the program. Need not signal a bug.
- **Bug** or **Mistake**: The program does the wrong thing.
- **Error-Handling Code**: Deals with a run-time error or exception.
- **Error-Handling Mistake**: A bug in your error-handling code.

## Where Do Run-Time Errors Come From?

- Exceptional situations come in two broad categories:
  - The “outside” world
  - Your “internal” language and APIs

## The Outside World

- Often exceptional situations derive from the outside world ...
- A mysterious area that **your program typically does not control**
- These situations are easy to overlook when programming
- And usually **must be handled** on a case-by-case basis ...

## “Outside” Exceptions

- A file is mistakenly deleted (or locked or has its permissions changed, ...)
- A disk drive runs out of space
- A network briefly disconnects (or no route to host, ...)
- A value read from a user (or the network, ...) is malformed
- A disk read or write fails (or the media is ejected in the middle, ...)
- Out of memory (or file handles, thread identifiers, ...)
- You don't have permission to do that
- The remote host died (or timed out, ...)

## “Internal” Problems

- It's also possible to get into trouble all by yourself
- Languages and virtual machines have low-level requirements about how your program should behave
- Usually these exceptions **indicate actual mistakes** in your program
- ... and thus you rarely “handle” them

## Thy Inward Part

- NULL pointer dereference
- Array Out Of Bounds (segfault)
- Floating Point Exception (div by 0, ...)
- Stack Overflow
- “You have already acquired that lock”
- Type mismatches (cast fails, ...)

## “At A Mortal War”

- Finally, you can fail to get along with your APIs
- Most APIs and libraries have preconditions or requirements (possibly implicit)
- Failing to respect these conditions is usually an **actual mistake in your program**
- ... and thus you rarely “handle” them

## Errors at Interfaces

- Passing -2 to sqrt()
- Passing “[ab]+[a” to regexp()
- Passing “23y4” to ascii\_to\_integer()
- Passing the same string to strtok() twice
- Trying to pop an empty stack
- Trying to write to an unopened socket
- Trying to write to a read-only file

## Where Do We Fit In?

- We're going to skip past interface errors and language-level errors
  - Those are basically bugs in your program and there is no real way to "handle" them
- And concentrate on environmental and third-party conditions
- What should we do while handling them?

## What Are Our Options?

- #1. Do Nothing
  - Do not try to report or handle errors, just die if something goes wrong
- #2. Return "Error Codes"
  - DOS 2.x - 18 total distinct codes
  - POSIX - 139 total distinct codes
  - DOS 3.x - one byte "extended" error codes
  - OS/2 - 16 bit error codes
  - NT - 16 bit error codes → 32 bit status codes
  - You must **check every function return**. Argh!

## More Options

- #3. Database Transactions
  - A set of actions (e.g., reads, writes) either all succeed (commit) or all fail (abort)
  - Pass the buck: **you talk to the database** and it does all of the error-handling for you (ironclad I/O, write-ahead logging, recovery, concurrency control, ...) but you must **handle errors the DB reports!**
  - Efforts to add transactions to programming languages directly have met with "limited success"
  - But: JDBC, Java Beans, Fortran, inline SQL, ...

## Our Hero

- #4. Exceptions & Exception Handling
  - Formalized by John Goodenough in '75
  - Now found in Java, C++, Ada, OCaml, C#, ...
- Vague Exception Idea:
  - Code can "signal" or "raise" an exception
  - And some other code "catches" and "handles" that exception
  - No need to check every return code
  - You may have heard this story before
    - So we'll look at some high-level exception concepts

## Exception Propagation?

- Initially, the immediate caller had to deal with the exception
  - In CLU if you call send() and you don't put an exception handler around it and it raises an error, the whole program aborts. The exception was **not propagated**.
  - Much like Java's (and Ada's) requirement that you say which exceptions your method can raise, this is often regarded as tedious.

## Implicit Propagation

- In Java, C++, C#, etc., when an exception is raised the run-time system **unwinds the call stack** until it finds an appropriate exception handler
- You can implement this sort of thing in C with setjmp() and longjmp(). Don't.
  - "setjmp() makes programs hard to understand and maintain. If possible an alternative should be used." - man setjmp

## How Can We Handle It?

- Most languages use the “replacement model” for exception handling:
  - If an exception is raised by a computation, the result of that computation is **replaced by the result of the nearest enclosing appropriate exception handler**.
- PL Theorists toy with the “resumption” model
  - In which you can handle an exception and then **return to the statement that caused it**, trying that statement again.
  - Not really found outside of Visual Basic, MSVC.

## Gory Details - Not Here!

- You should understand how exceptions are implemented so that you don't make the **mistake of thinking they are too expensive** to use in your program.
- But I won't explain that here. Basically:
  - “try” is free
  - “finally” is cheap
  - “throw” and “raise” are expensive

## Exception Summary

- Wary code (like yours!) installs an “**exception handler**” (try/catch) before calling a subroutine
- The subroutine calls another method that eventually **throws** (signals, raises) an exception
- The run-time system peels methods off of the call stack, executing any “**finally**” blocks it encounters, until it finds an appropriate handler
- That **handler executes** and control continues on from the end of that handler (not from the source of the exception!)

## Exception Handling Example

- I'm sure you've seen it before, but for the benefit of anyone who has not seen this syntax ... (C++)

```
#include <exception>
String SlurpWebsiteIndex(String website) {
    try {
        // dangerous activities: play with sockets, etc.
    } catch (std::exception &exc) {
        // handle the error somehow
    } catch (...) {
        // handle all other errors somehow
    }
}
```

## Exception Handling Example

- Concrete Java example:

```
String SlurpWebsiteIndex(String website)
    throws IOException {
    Socket s = new Socket(website,80);
    try {
        s.send("GET index.html\n");
        return s.readWholeThing();
    } catch (Exception e) { // handle all errors
        System.out.println("It didn't work! Reason: " + e);
        return NULL;
    } finally {
        s.close();
    }
}
```

## Software Engineering Note

- Exceptions can break encapsulation!
- If your Stack class can throw an ArrayOutOfBoundsException, you are giving away **implementation details**
- At interface boundaries it is prudent to catch all exceptions and “rethrow” them with interface-level descriptions of the problem ... make sure you don't hide too much information!

## Exception Hints

- Never catch “any kind of exception” without rethrowing it
  - You may be **masking** your caller’s handler for a kind of exception you have not foreseen
- Reduce code size: write a few procedures for handling common situations (if possible) and call them from handlers
- **Document** every empty handler you write

## Scylla and Charybdis

- The primary useful property of exceptions is the **non-local control flow**: jump to the nearest enclosing handler
- The most dangerous property of exceptions is the **non-local control flow**: jump to the nearest enclosing handler

## C++ Example

- From Cargill’s “Exception Handling: A False Sense of Security” - a stack copy constructor that should be “exception neutral”

```
template Stack::Stack(const Stack& s) {  
    v = new T[nelems = s.nelems];  
    if ( v == 0 ) throw “out of memory”;  
    if ( s.top > -1 ) {  
        for (top = 0; top <= s.top; top++)  
            v[top] = s.v[top];  
        top--;  
    }  
}
```

Can anything go wrong?

## C++ Example

- From Cargill’s “Exception Handling: A False Sense of Security” - a stack copy constructor that should be “exception neutral”

```
template Stack::Stack(const Stack& s) {  
    v = new T[nelems = s.nelems]; // memory leak  
    if ( v == 0 ) throw “out of memory”;  
    if ( s.top > -1 ) {  
        for (top = 0; top <= s.top; top++)  
            v[top] = s.v[top]; // throw  
        top--;  
    }  
}
```

## Ouch!

- It would be unfortunate if our desire to handle errors or support error handling actually helped us to make a mistake (e.g., leak a resource)
- Exceptions mean that almost any statement in your method can “jump out” to a handler rather than continuing on to your next statement

## Taking Care

- Exceptions introduce **hard-to-see control flow paths** that may skip over your code
- Thus if you ever “have to do something later” you must do it in an exception-aware way
- Even your error-handling code can have errors!

## The State Of The Art

- Most Common Exception Handlers
  - #1. Do Nothing
  - #2. Print Stack Trace, Abort Program
- Higher-level invariants should be restored, interface requirements should be respected
  - Aside from handling the error, the **code should clean up after itself**

## What's Up In Real Life?

- It is difficult for programmers to consider all of the possible execution paths in the presence of exceptions
- So there are often a few **paths related to exceptional conditions** in which the error handling does not work as well as it should
- Let's see an example:

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input =
        new BufferedInputStream(sourceURL.openStream());
    BufferedOutputStream output =
        new BufferedOutputStream(
            new FileOutputStream(destinationFile));
    // The resource pointed to might be a pdf file, which
    // is binary, so we are careful to read it byte by
    // byte and not do any conversions of the bytes.
    int c;
    while ((c = input.read()) != -1) {
        output.write(c);
    }
    output.close();
    input.close();
} // line 294
```

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input =
        new BufferedInputStream(sourceURL.openStream());
    BufferedOutputStream output =
        new BufferedOutputStream(
            new FileOutputStream(destinationFile));
    // The resource pointed to might be a pdf file, which
    // is binary, so we are careful to read it byte by
    // byte and not do any conversions of the bytes.
    int c;
    while ((c = input.read()) != -1) {
        output.write(c);
    }
    output.close();
    input.close();
} // line 294
```

Can anything go wrong?

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input =
        new BufferedInputStream(sourceURL.openStream());
    BufferedOutputStream output =
        new BufferedOutputStream(
            new FileOutputStream(destinationFile));
    // The resource pointed to might be a pdf file, which
    // is binary, so we are careful to read it byte by
    // byte and not do any conversions of the bytes.
    int c;
    while ((c = input.read()) != -1) {
        output.write(c);
    }
    output.close();
    input.close();
} // line 294
```

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input; BufferedOutputStream output;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));
        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        output.close();
        input.close();
    }
} // line 294
```

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input; BufferedOutputStream output;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));

        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        output.close();
        input.close();
    }
} // line 294
```

Can anything go wrong?

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input; BufferedOutputStream output;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));
        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        output.close();
        input.close();
    }
} // line 294
```

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input = null;
    BufferedOutputStream output = null;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));

        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        if (output != null) output.close();
        if (input != null) input.close();
    }
} // line 294
```

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input = null;
    BufferedOutputStream output = null;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));

        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        if (output != null) output.close();
        if (input != null) input.close();
    }
} // line 294
```

Can anything go wrong?

## Java Exception Example

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input = null;
    BufferedOutputStream output = null;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));

        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        if (output != null) output.close();
        if (input != null) input.close();
    }
} // line 294
```

When could output.close() raise an exception?

## Fixed Exception Example #1

```
// Copy sourceURL to destinationFile without doing any byte conversion.
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input = null;
    BufferedOutputStream output = null;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        output = new BufferedOutputStream(
            new FileOutputStream(destinationFile));

        int c;
        while ((c = input.read()) != -1) {
            output.write(c);
        }
    } finally {
        if (output != null) try { output.close(); } catch (Exception e) {}
        if (input != null) try { input.close(); } catch (Exception e) {}
    }
} // line 294
```

## Fixed Exception Example #2

```
private static void _binaryCopyURLToFile(URL sourceURL,
File destinationFile) throws IOException {
    BufferedInputStream input; BufferedOutputStream output;
    try {
        input = new BufferedInputStream(sourceURL.openStream());
        try {
            output = new BufferedOutputStream(
                new FileOutputStream(destinationFile));
            int c;
            while ((c = input.read()) != -1) {
                output.write(c);
            }
        } finally {
            try { output.close(); } catch (Exception e) {}
        }
    } finally {
        try { input.close(); } catch (Exception e) {}
    }
} // line 294
```

## What next?

- That looks like a big problem here on the screen, but does it happen in real life?
- I claim: **Yes!**
- And I'll present analysis results to back that up.
- Just having exception handling code is not enough - that code should be correct as well.

## Cunning Plan

- We want to show that programs make mistakes in their run-time error handling
- **To show a mistake**, we need to know what the program should have been doing
- So we surveyed 4 million lines of Java code (catch, finally, finalize)
- And found four main things that programmers tried to do in them

## The Four Horsemen

- Four important (Java) resources
  - **Sockets**
  - **Streams**
  - **File Handles**
  - **JDBC Database Objects**
- Safety policy:
  - If you acquire one of these you should eventually release it, even if you have to handle an exceptional situation!

## Making the Exceptional Mundane

- How do you (write a) test for exceptional conditions?
- Usual answer: you don't ...
  - And thus the "coverage" of your test suite is rarely 100%
- Unusual answer: ???
  - **Any ideas?**

## Making the Exceptional Mundane

- How do you (write a) test for exceptional conditions?
- Usual answer: you don't ...
  - And thus the "coverage" of your test suite is rarely 100%
- Unusual answer: fault injection
  - **Intentionally simulate faults** to see how your program will react

## Fault Injection

- Ad Hoc Approach:
  - Pull plug while program is running
- Systematic Approach, the gist of it:
  - Since your program does not use I/O (etc.) directly but instead talks through layers and wrappers (right?) you can easily **change your wrapper** around send() or open() to fail every 10<sup>th</sup> time (or every *n*<sup>th</sup> time, etc.)

## We Won't Do That, Exactly

- ... because it requires that you:
  - #1. Be able to run the program.
  - #2. Have a good test suite.
  - #3. Have good encapsulation and a good test harness.
- If we want to show that this problem is common in real life, we have to pull programs off the shelf ...

## Static Fault Injection

- It is also possible to examine the program **statically** (like a dataflow analysis or a type checker) and consider what would happen if an exceptional event occurred
- Rather than having every 10<sup>th</sup> send() fail, for every possible source of failure we consider what will happen if it fails and what will happen if it succeeds

## Fault Model

- When a method is invoked it can **either** terminate normally **or** raise any of its declared exceptions.
  - Benefit of the Doubt: If we don't have the signature for a method it can either terminate normally or raise any exception for which there is a lexically enclosing catch block (usually there are none).

## Example Exception Signature

```
/**
 * Connects the socket with a remote address. A timeout of zero is interpreted as
 * an infinite timeout. The connection will then block until established or an error occurs.
 *
 * @param endpoint The address to connect to
 *
 * @exception IOException If an error occurs
 * @exception IllegalArgumentException If the address type is not supported
 * @exception IllegalBlockingModeException If this socket has an associated channel, and
 * the channel is in non-blocking mode
 * @exception SocketTimeoutException If the timeout is reached
 *
 * @since 1.4
 */
public void connect(SocketAddress endpoint, int timeout) throws IOException {
    if (closed) throw new SocketException ("Socket is closed");

    if (! (endpoint instanceof InetSocketAddress))
        throw new IllegalArgumentException ("Address type not supported");

    if (ch != null && !ch.isBlocking ()) throw new IllegalBlockingModeException ();

    impl.connect (endpoint, timeout);
} /* from: java/net/Socket.java */
```

What throws the SocketTimeoutException?

## Analysis Summary

- Parse program text, obtain syntax tree
- Build the control-flow graph
  - Holy CS164, Batman!
  - But pay special attention to exceptional control flow.
- **Symbolically execute each method**
  - Simulate: pretend you are the CPU or JVM
  - But ignore most data values
  - At branches, simulate both directions

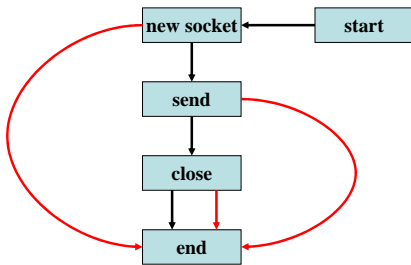
## Analysis Computation

- For each path through the CFG
  - Track all outstanding resources
  - Abstract away data values
- Process one method at a time
  - Intraprocedural
- Find an error **path** through the method where a resource is leaked
- Let's see an example ...

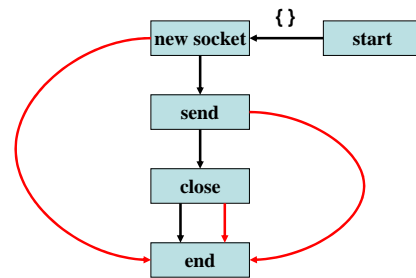
## My Favorite Toy Story

```
try {  
  Socket s = new Socket();  
  s.send("GET index.html");  
  s.close();  
} finally { }  
// close should be in finally
```

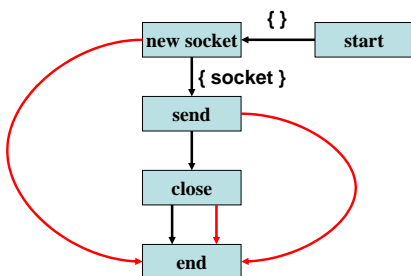
## Analysis Example



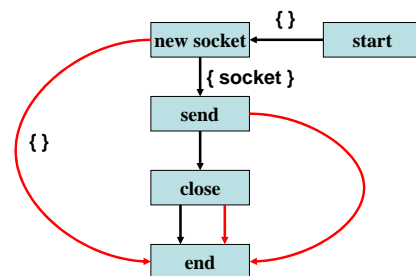
## Analysis Example

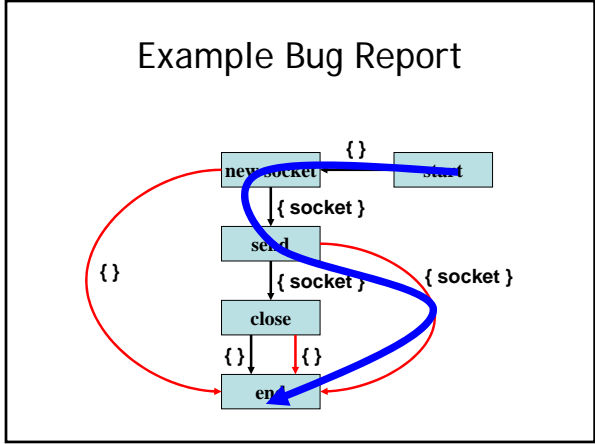
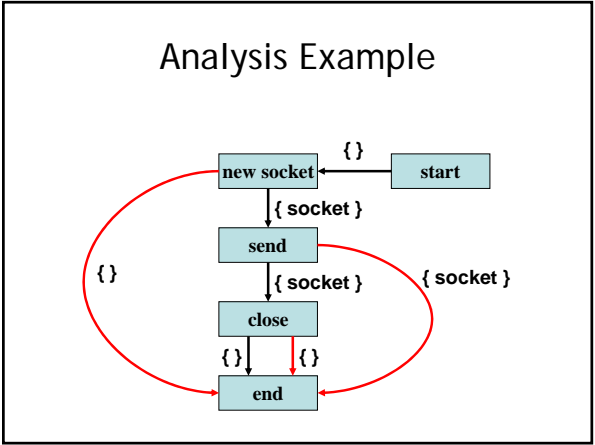
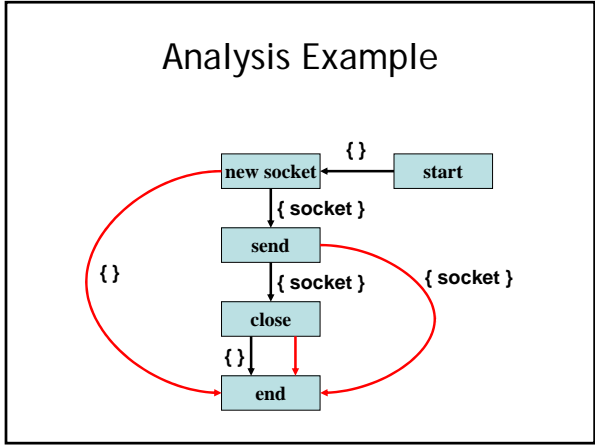
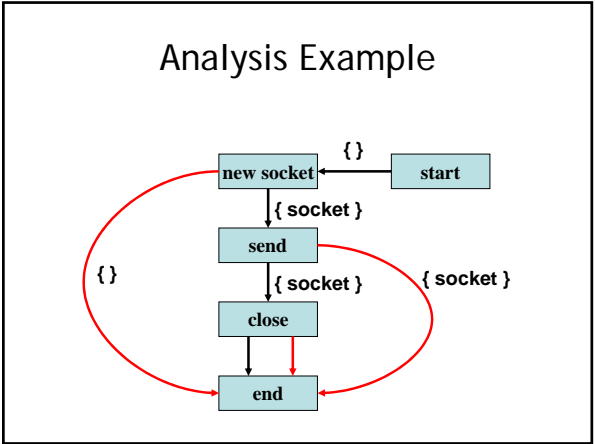
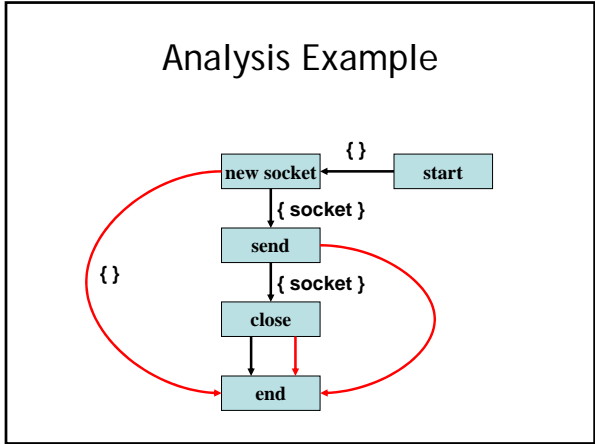


## Analysis Example



## Analysis Example





- ### What Can Go Wrong?
- What do **you**, the viewers at home, think?
  - How might this analysis not give us useful information?

## False Positives

- Simple analysis gives **false positives**
  - Of 100 error reports
  - 70 would be real bugs
  - 30 would be false positives
- Each false positive is one of these:
  - if (sock != null) sock.close();
  - socket\_field\_we\_close\_later = sock;
  - return sock;

## Bug Reporting and Filtering

- Filter out reports with those 3 forms
  - Introduces false negatives
  - For every 100 real bugs reported
  - There are ~10 real bugs we miss
- Removes all false positives
  - Simple to understand rules
- Manually verify the rest as bugs

## Analysis Results

Program Name	Lines of Code	Mistakes with Error Handling Missing	Forgotten Resources
hl bernate2	57k	13	DB
j axme	58k	4	File
axl on	65k	16	File
hsqj db	71k	18	DB, Strm
cayenne	86k	7	File
sabl ecc	99k	3	Strm
j boss	107k	40	DB, Strm
mckol -sql	118k	27	Strm, DB
portal	162k	98	DB, File
pcgen	178k	17	File
compl ere	230k	310	DB
org. aspectj	319k	27	File, Strm
ptol emy2	362k	27	File, Strm
ecl i pse	1.6M	126	Strm, File
... 13 others ...	347k	141	File, DB
<b>Total</b>	<b>3.9M</b>	<b>618</b>	<b>File, DB</b>

## Characterizing Mistakes

- Very Common: **No Error Handling At All**
  - axion's ObjectBTree class:

```
public void read() throws IOException, /* ... */ {
    File idxFile = getFileById(getFileId());
    // ...
    FileInputStream fin = new FileInputStream(idxFile);
    ObjectInputStream in = new ObjectInputStream(fin);
    // ...
    in.close();
    fin.close();
}
```

## Characterizing Mistakes (#2)

- Common: **Protect Some Areas, But Not All**
  - staf's STAXMonitor class:

```
ObjectInputStream ois = null;
try {
    ois = new ObjectInputStream(/* ... */);
    // ...
} catch (StreamCorruptedException ex) {
    if (ois != null) { ois.close(); }
    showErrorDialog(/* ... */);
    return false;
}
Object obj = ois.readObject(); // no try
ois.close(); // no finally
```

## Characterizing Mistakes (#2)

- Common: **Protect Some Areas, But Not All**
  - staf's STAXMonitor class:

```
ObjectInputStream ois = null;
try {
    ois = new ObjectInputStream(/* ... */);
    // ...
} catch (StreamCorruptedException ex) {
    if (ois != null) { ois.close(); }
    showErrorDialog(/* ... */);
    return false;
}
Object obj = ois.readObject(); // no try
ois.close(); // no finally
```

Why would a program ever end up only protecting some things?

## Your Turn: Find The Bug

```
// com/ohioedge/j2ee/api/org/tool/ejb/LetterTemplateEJB.java, line 154
private StringBuffer getColumnStringBufferFromReader()
{
    StringBuffer buf = null;
    java.io.Reader reader = null;
    PreparedStatement prepStmt = null;
    ResultSet rs = null;
    Connection cn = null;
    try {
        cn = ConnectionFactory.getConnection("jdbc:LetterTemplateDB");
        StringBuffer qry = new StringBuffer();
        qry.append(" SELECT \"LetterTemplate\" FROM LetterTemplate ");
        qry.append(" WHERE \"LetterTemplateID\" = "+getLetterTemplateID());
        prepStmt = cn.prepareStatement(qry.toString());
        rs = prepStmt.executeQuery();
        while (rs.next())
            reader = rs.getCharacterStream(1);
        buf = readIntoStringBuffer(reader);
        rs.close();
        prepStmt.close();
    } catch (Exception e) {
        log.error(this.getClass().getName()+" getColumnStringBufferFromReader():");
        e.printStackTrace();
    } finally {
        try {
            cn.close();
        } catch (Exception e1) {
            log.error(this.getClass().getName()+" getColumnStringBufferFromReader():");
            e1.printStackTrace();
        }
    }
    return buf;
}
```

Note error handler that prints a stack trace and does nothing else ...

## Characterizing Mistakes

- Model: a1 must be followed by c1, etc.
  - e.g., a1 = getLock(), c1 = releaseLock()
- What is wrong with:
  - 1) try { a1; } finally { c1; }; a1; c1;
  - 2) try { a1; a2; } finally { c2; c1; }
  - 3) try { a1; a1; } finally { c1; }
  - 4) for (...) { a1; work; c1; }

## Destructors (C++, C#, ...)

- Great for **stack-allocated** objects
- Error-handling contains arbitrary code
  - Example adapted from code which has 17 unique cleanups, one 34 lines long

```
try {
    StartDate = new Date(); // a1
    try {
        StartLSN = log.getLastLSN(); // a2
        work(1);
        try {
            DB.getWriteLock(); // a3
            work(2);
        } finally {
            DB.releaseWriteLock(); // c3
            work(3); // c3
        } finally {
            StartLSN = -1; // c2
        } finally {
            StartDate = null; // c1
        }
    }
}
```

## Finalizers (Java, ...)

- Called by the garbage collector
  - **Too late!**
- No ordering guarantees
  - Socket sock = new Socket();
  - Stream strm = new Stream(sock);
  - sock.finalize may be called before strm.finalize
- Programs **do not use** them
  - Only 13 user-defined ones in 4M LOC
  - Not all SDKs use them for key resources

## Finalizer Bonus Trick

- Use finalizers to detect and **track leaks!**
- ```
Class myRes {
    int index;
    String callsite;
    bool closed;
    myRes(String callsite) {
        closed = false;
        ...
    }
    void close() {
        closed = true;
    }
    void finalize() {
        if (!closed) yell(index, ...);
    }
}
```

## Fix It With Flags

```
int f = 0; // flag tracks progress
try {
    openX(); f = 1; work();
    openY(); f = 2; work();
    openZ(); f = 3; work();
} finally {
    switch (f) { // note fall-through!
        case 3: try { closeZ(); } catch (Exception e) {}
        case 2: try { closeY(); } catch (Exception e) {}
        case 1: try { closeX(); } catch (Exception e) {}
    }
}
```

## Fix It With Flags

```
int f = 0;           // flag tracks progress
try {
    openX(); f = 1; work();
    openY(); f = 2; work();
    openZ(); f = 3; work();
} finally {
    switch (f) {      // note fall-through!
        case 3: try { closeZ(); } catch (Exception e) {}
        case 2: try { closeY(); } catch (Exception e) {}
        case 1: try { closeX(); } catch (Exception e) {}
    }
}
```

What are some drawbacks to this error-handling approach?

## What To Do?

- I have a research solution
  - But I won't recommend it to you. :-)
- Instead:
  - Use destructors if you have them
  - Build something similar if you don't
  - Use try/finally around important resources
  - Use nested try/finally or checks against NULL
  - Use finalizers to track resource leaks
- Exceptions add hidden control flow!

## Further Reading

- <http://www.gnu.org/software/gdb/documentation/> - your project might benefit from something like gdb's "cleanup chains". Look them up.
- [http://www.awprofessional.com/content/images/020163371x/supplements/Exception\\_Handling\\_Article.html](http://www.awprofessional.com/content/images/020163371x/supplements/Exception_Handling_Article.html) - Cargill's article about C++ exception woes.

## Transactional Reading

- [http://research.compaq.com/wrl/people/dlowell/papers/dc\\_tech\\_report.pdf](http://research.compaq.com/wrl/people/dlowell/papers/dc_tech_report.pdf) - Lowell's "Discount Checking" gives low-overhead (0.6%) failure transparency
- <http://research.compaq.com/wrl/people/dlowell/papers/osdi00.pdf> - Lowell's "Exploring Failure Transparency and the Limits of Recoverability" explains why checkpointing isn't everything.

## Exceptional Reading

- <http://portal.acm.org/citation.cfm?id=361230> - Goodenough's seminal work on formalizing exception handling
- <http://citeseer.ist.psu.edu/robillard99regaining.html> - "Regaining Control of Exception Handling" talks about why the structure of Java exception handling degrades over time

## Read and Pull the Plug

- <http://portal.acm.org/citation.cfm?id=837386> - "Automatic Failure Path Inference" - See Section 3.1 for how real life faults (e.g., database access errors, "bad server in registry") map to Java exceptions. They present a technique for discovering the "failure dependencies" of your program.

## Read and Test

- <http://prolang.s.rutgers.edu/refs/docs/dcs-tr-545.pdf> - "Testing of Java Web Services for Robustness" - describes a white-box technique for coverage testing of error handling code. An analysis directs fault injection via a test harness. Consider doing something similar when you make your test suite.

## Read Charts and Graphs

- [http://www.nist.gov/public\\_affairs/releases/n02-10.htm](http://www.nist.gov/public_affairs/releases/n02-10.htm) - Software Errors Cost U.S. Economy \$59.5 Billion Annually
- <http://portal.acm.org/citation.cfm?id=1028976.1029011> - "Finding And Preventing Run-Time Error Handling Mistakes" - the research behind the second bit of this presentation, plus a proposed Java language feature to do this work for you

## Conclusion

- Error handling is important and significant
  - \$ cost, reliability, security, code size
- Run-time errors and exceptions are used to signal
  - API violations (send() on closed socket)
  - Environment conditions (congested network, no fds)
- Error handling code
  - Handles the error (resend packet, display window)
  - Does not leak any resources
- Handle with care
  - Destructors, Finalizers, try/catch/finally
- Exceptions introduce new control-flow paths
  - And if you don't consider them, you may introduce mistakes

## Any Questions?