Java Exception Handling

Handling errors using Java's exception handling mechanism

Approaches For Dealing With Error Conditions

- Use branches/decision making and return values
- Use Java's exception handling mechanism

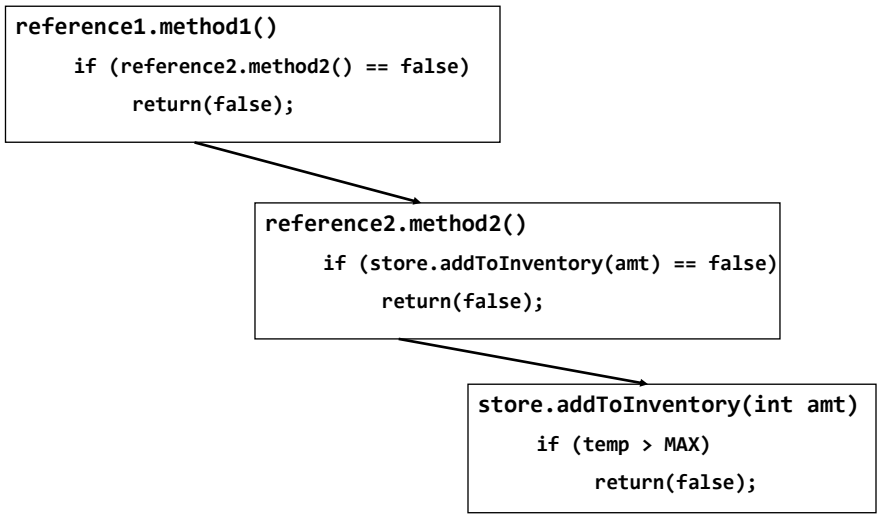
Class Inventory: An Earlier Example

```
public class Inventory
{
    public final int MIN = 0;
    public     final int MAX = 100;
    public final int CRITICAL = 10;
    public boolean add(int amount)
    {
       int temp;
        temp = stockLevel + amount;
        if (temp > MAX)
        {
            System.out.print("Adding " + amount + " item will
                              cause stock ");
            System.out.println("to become greater than " + MAX +
                               " units (overstock)");
            return(false);
        }
```

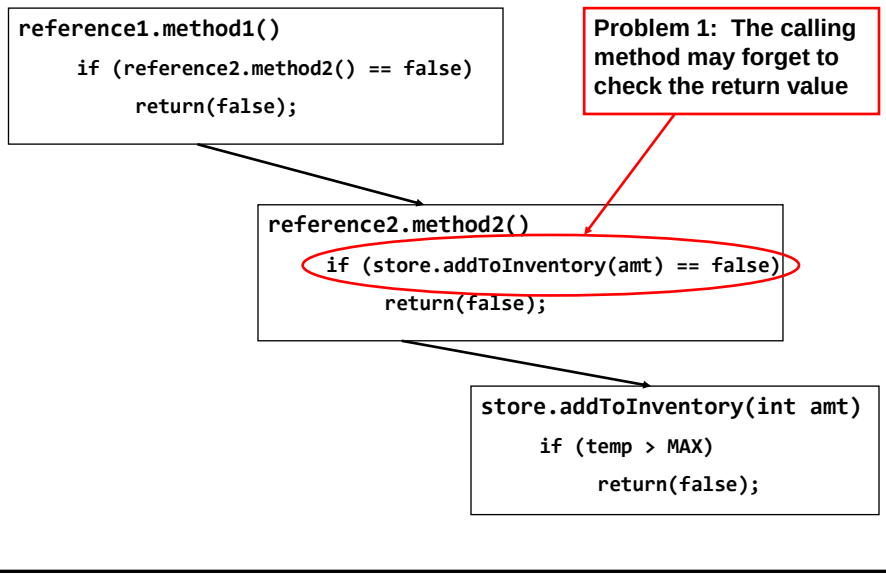
Class Inventory: An Earlier Example (2)

```
        else
       {
            stockLevel = stockLevel + amount;
           return(true);
        }
    } // End of method add()
    ...
```

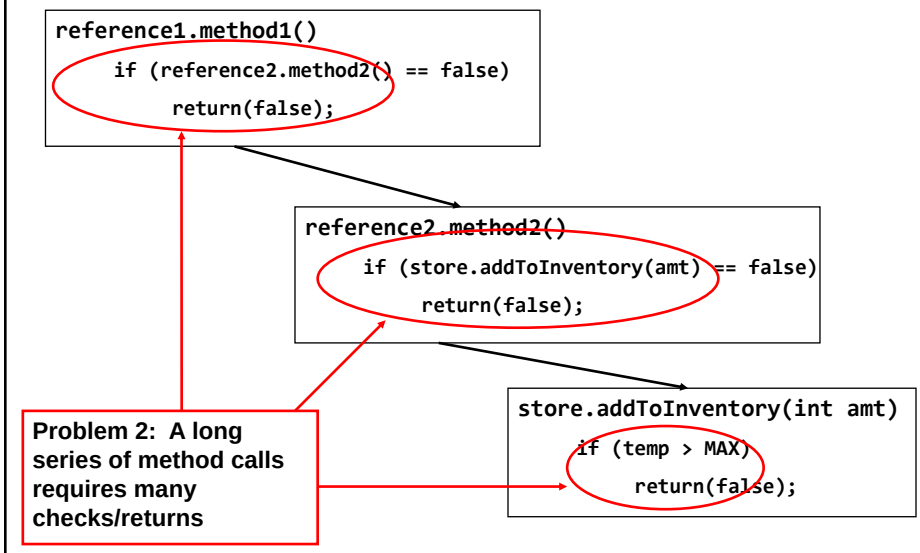
Some Hypothetical Method Calls: Condition/Return



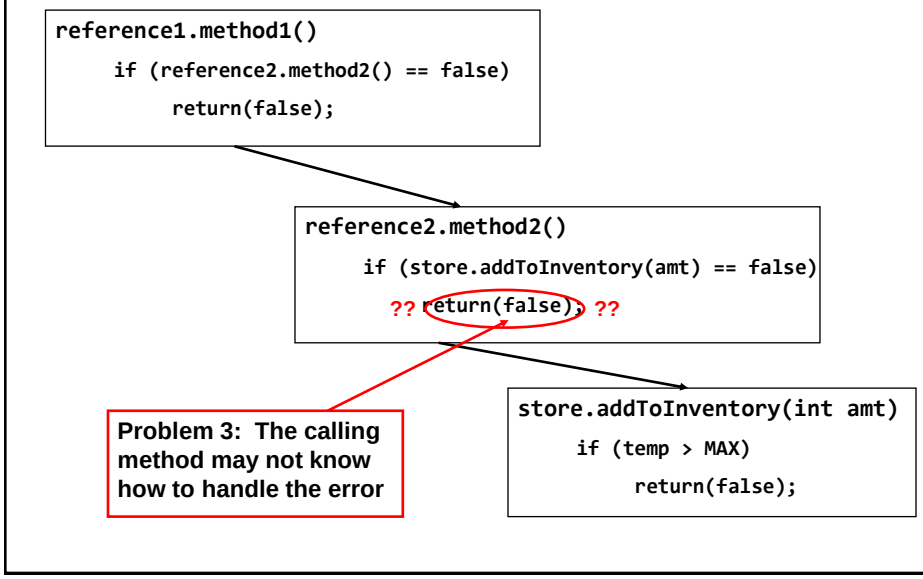
Some Hypothetical Method Calls: Condition/Return



Some Hypothetical Method Calls: Condition/Return



Some Hypothetical Method Calls: Condition/Return



Approaches For Dealing With Error Conditions

- Use branches/decision making constructs and return values
- Use Java's exception handling mechanism

Handling Exceptions

Format:

```
try
{
    // Code that may cause an error/exception to occur
}
catch (ExceptionType identifier)
{
    // Code to handle the exception
}
```

Handling Exceptions: Reading Input

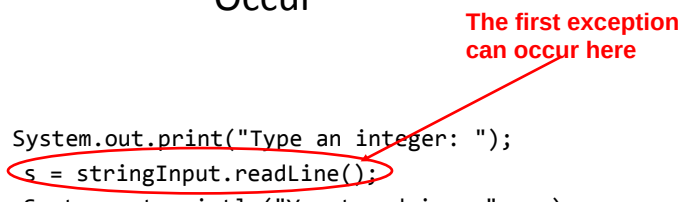
Location of the online example:
/home/219/examples/exceptions/handlingExceptions/inputExample

```
public class Driver {
    public static void main (String [] args)
    {
        BufferedReader stringInput;
        InputStreamReader characterInput;
        String s;
        int num;
        characterInput = new InputStreamReader(System.in);
        stringInput = new BufferedReader(characterInput);
```

Handling Exceptions: Reading Input (2)

```
        try
        {
            System.out.print("Type an integer: ");
            s = stringInput.readLine();
            System.out.println("You typed in..." + s);
            num = Integer.parseInt (s);
            System.out.println("Converted to an integer..."
                               + num);
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
        catch (NumberFormatException e)
        {
            ...
        }
    }
}
```

Handling Exceptions: Where The Exceptions Occur




```
try
{
      System.out.print("Type an integer: ");
      s = stringInput.readLine();
       System.out.println("You typed in..." + s);
       num = Integer.parseInt (s);
       System.out.println("Converted to an integer..."
                          + num);
 }
```

Handling Exceptions: Result Of Calling BufferedReader.ReadLine()

```
try
{
      System.out.print("Type an integer: ");
       s = stringInput.readLine();
       System.out.println("You typed in..." + s);
       num = Integer.parseInt (s);
       System.out.println("Converted to an integer..."
                          + num);
 }
```

Where The Exceptions Occur In Class BufferedReader

- For online documentation for this class go to:
  - http://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html

```
public class BufferedReader
{
    public BufferedReader(Reader in);
    public BufferedReader(Reader in, int sz);
    public String readLine() throws IOException;
      ...
}
```

Handling Exceptions: Result Of Calling Integer.ParseInt ()

```
try
{
     System.out.print("Type an integer: ");
      s = stringInput.readLine();
      System.out.println("You typed in..." + s);
      num = Integer.parseInt (s);
      System.out.println("Converted to an integer..."
                          + num);
 }
```

The second exception can occur here

Where The Exceptions Occur In Class `Integer`

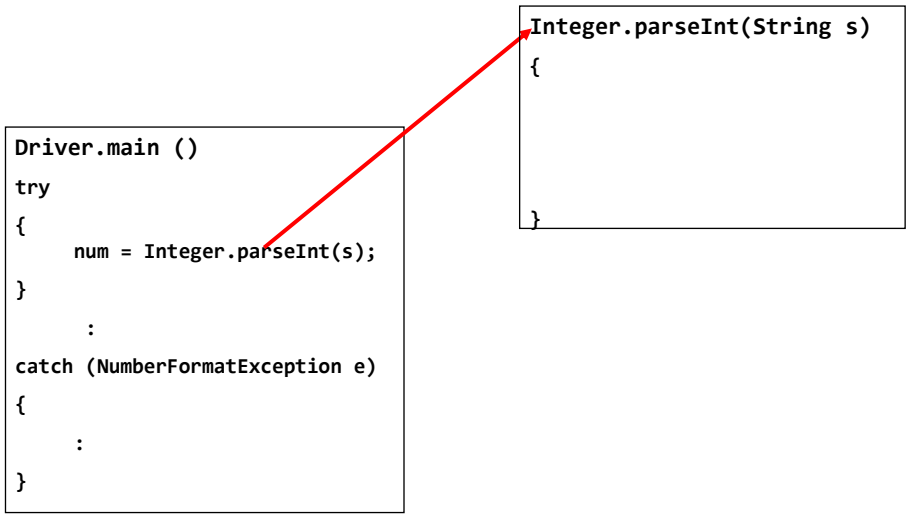
- For online documentation for this class go to:
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

```
public class Integer
{
    public Integer(int value);
    public Integer(String s) throws NumberFormatException;
      ...
    public static int parseInt(String s) throws
                                NumberFormatException;
      ...
}
```

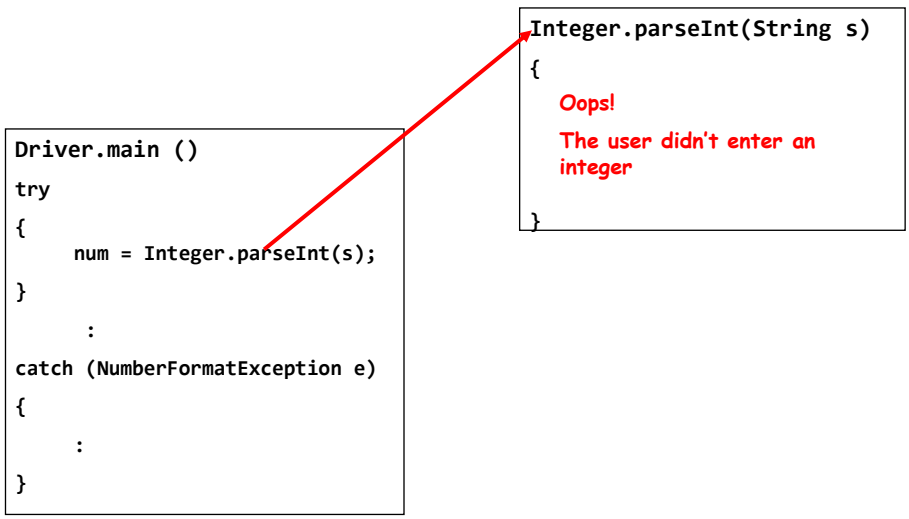
Handling Exceptions: The Details

```
        try
        {
            System.out.print("Type an integer: ");
            s = stringInput.readLine();
            System.out.println("You typed in..." + s);
            num = Integer.parseInt (s);
            System.out.println("Converted to an integer..."
                               + num);
         }
         catch (IOException e)
         {
            System.out.println(e);
         }
         catch (NumberFormatException e)
         {
             ...
         }
    }
}
```

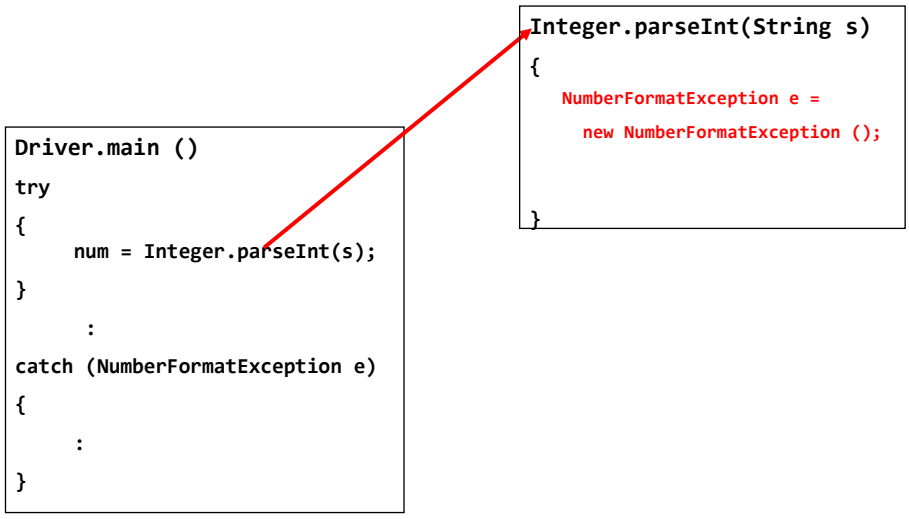
Handling Exceptions: Tracing The Example



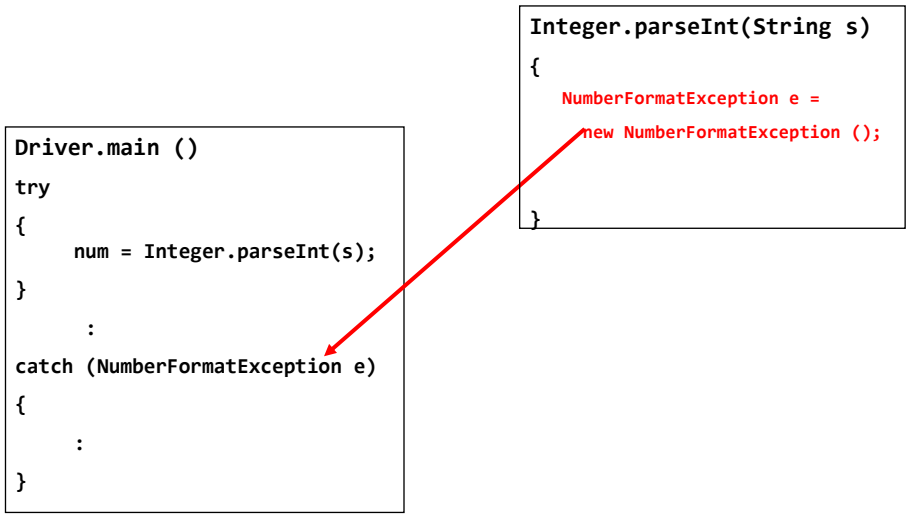
Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example

```
Integer.parseInt(String s)
{
   NumberFormatException e =
     new NumberFormatException ();


}
```

```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
  Exception must be dealt
  with here
}
```

Handling Exceptions: Catching The Exception

```
        catch (NumberFormatException e)
        {
            ...
        }
    }
}
```

Catching The Exception: Error Messages

```
        catch (NumberFormatException e)
        {
            System.out.println("You entered a non-integer
                                value.");
            System.out.println(e.getMessage());
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Catching The Exception: Error Messages

```
        catch (NumberFormatException e)
        {
            System.out.println("You entered a non-integer
                                value.");
            System.out.println(e.getMessage());
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

For input string: "james tam"

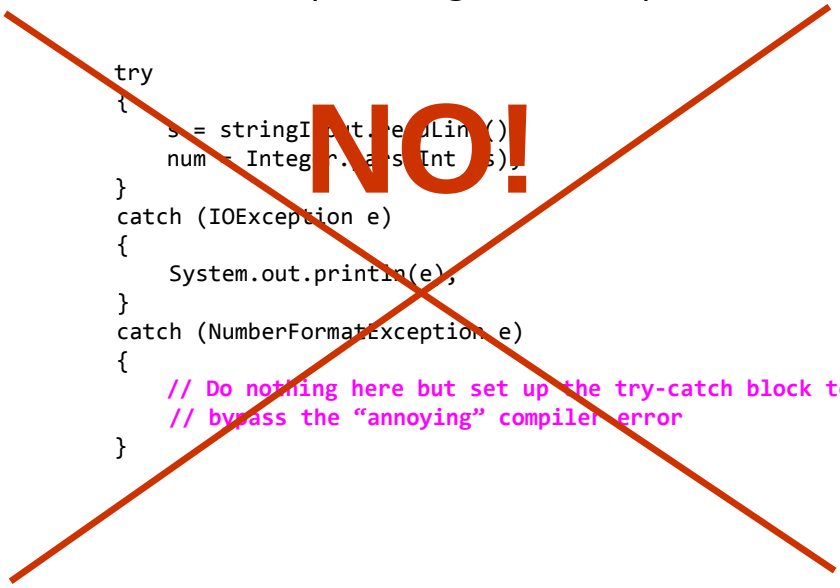
java.lang.NumberFormatException:
For input string: "james tam"

```
java.lang.NumberFormatException: For input string: "james tam"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:426)
    at java.lang.Integer.parseInt(Integer.java:476)
    at Driver.main(Driver.java:39)
```

Avoid Squelching Your Exceptions

```
try
{
    s = stringInput.readLine();
    num = Integer.parseInt (s);
}
catch (IOException e)
{
    System.out.println(e);
}
catch (NumberFormatException e)
{
    // Do nothing here but set up the try-catch block to
    // bypass the "annoying" compiler error
}
```

Avoid Squelching Your Exceptions




```
try
{
    s = stringInput.readLine();
    num = Integer.parseInt (s);
}
catch (IOException e)
{
    System.out.println(e);
}
catch (NumberFormatException e)
{
    // Do nothing here but set up the try-catch block to
    // bypass the "annoying" compiler error
}
```

Avoid Squelching Your Exceptions

```
try
{
    s = stringInput.readLine();
    num = Integer.parseInt (s);
}
catch (IOException e)
{
   System.out.println(e);
}
catch (NumberFormatException e)
{
    // Minimal but still somewhat useful response
    System.out.println("A non integer value entered
                        instead of an integer");
}
```

The `Finally` Clause

- An additional part of Java's exception handling model (`try-catch-`*`finally`*).
- Used to enclose statements that must always be executed whether or not an exception occurs.

The `Finally` Clause: Exception Thrown

```
try
{
    f.method();
}
```

```
catch
{
}
```

```
finally
{
}
```

```
f.method ()
{


}
```

The `Finally` Clause: Exception Thrown

```
try
{
    f.method();
}
```

1) Attempt to execute the method in the try block that may throw an exception

```
f.method ()
{
}
```

2) Exception thrown here

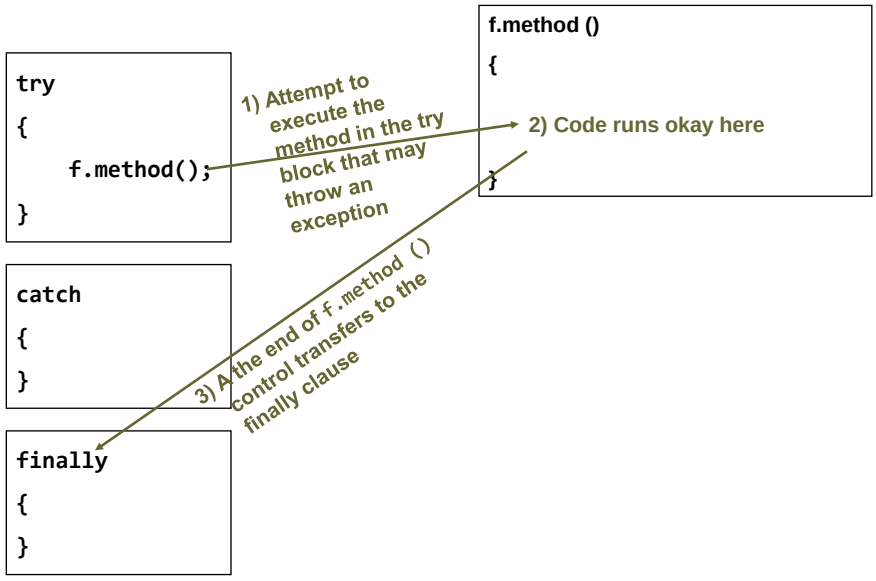
3) Exception is caught here

```
catch
{
}
```

4) A the end of the catch block control transfers to the finally clause

```
finally
{
}
```

The `Finally` Clause: No Exception Thrown



`Try-Catch-Finally`: An Example

Location of the online example:
`/home/219/examples/exceptions/handlingExceptions/tryCatchFinallyExample`

```
public class Driver
{
    public static void main (String [] args)
    {
        TCFExample eg = new TCFExample ();
        eg.method();
    }
}
```

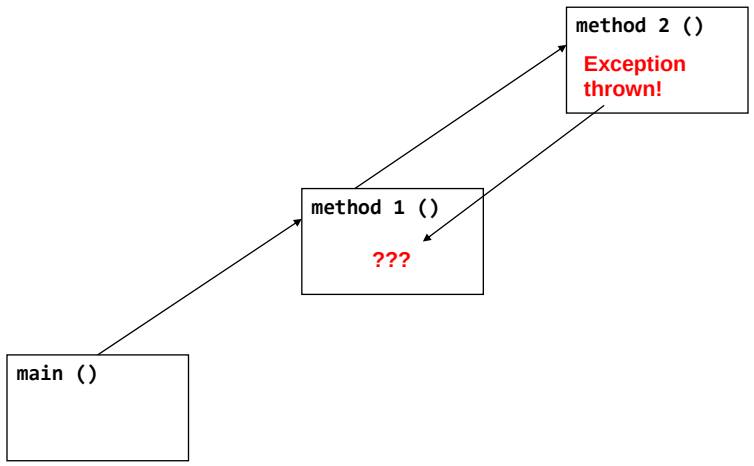
Try-Catch-Finally: An Example (2)

```
public class TCFExample
{
  public void method ()
    {
        BufferedReader br;
        String s;
        int num;
        try
        {
            System.out.print("Type in an integer: ");
            br = new BufferedReader(new
                   InputStreamReader(System.in));
            s = br.readLine();
            num = Integer.parseInt(s);
            return;
        }
```

Try-Catch-Finally: An Example (3)

```
        catch (IOException e)
        {
            e.printStackTrace();
            return();
        }
        catch (NumberFormatException e)
        {
            e.printStackTrace ();
            return();
        }
        finally
        {
            System.out.println("<<<This code will always
                                execute>>>");
            return;
        }
    }
}
```

When The Caller Can't Handle The Exceptions



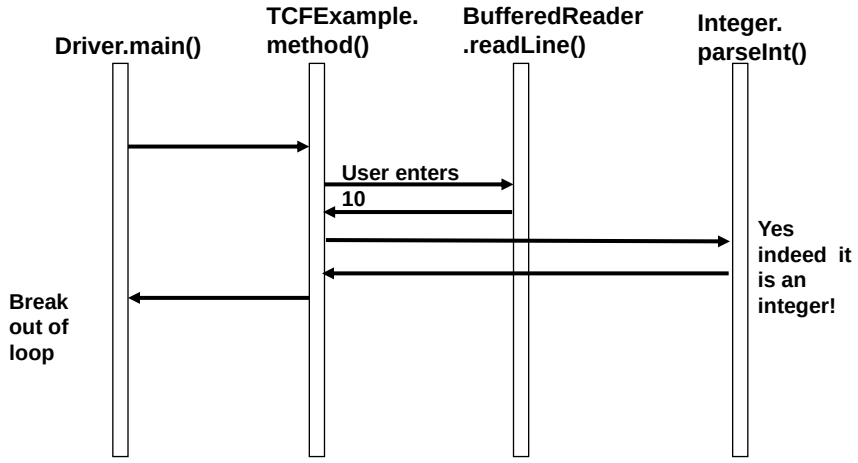
When The Caller Can't Handle The Exceptions: An Example

Location of the online example:

`/home/219/examples/exceptions/handlingExceptions/delegatingExceptions`

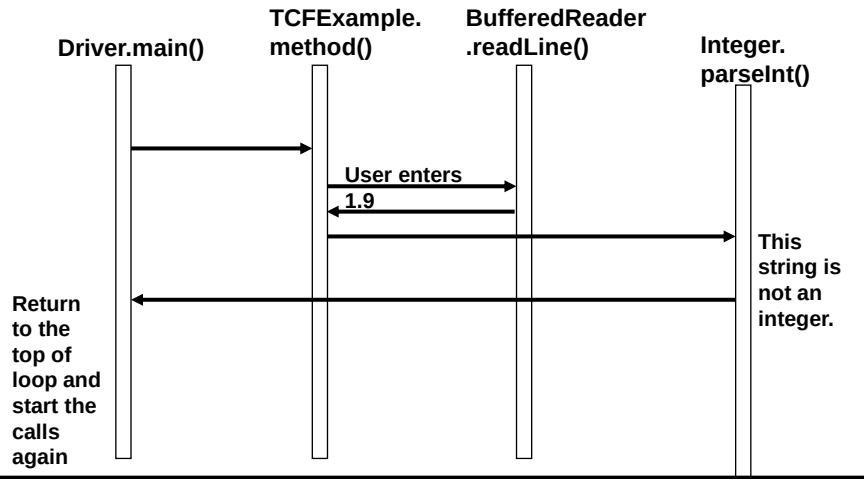
When The Caller Can't Handle The Exceptions: An Example (2)

•Tracing the method calls when *no exception occurs*:



When The Caller Can't Handle The Exceptions: An Example (3)

•Tracing the method calls when an *exception does occur*:



When The Caller Can't Handle The Exceptions: An Example (4)

```
public class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        boolean inputOkay = true;
```

When The Caller Can't Handle The Exceptions: An Example (5)

```
        do {
            try {
                eg.method();
                inputOkay = true;
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            catch (NumberFormatException e) {
                inputOkay = false;
                System.out.println("Please enter a whole
                                    number.");
            }
        } while(inputOkay == false);
    }  // End of main
} // End of Driver class
```

When The Caller Can't Handle The Exceptions: An Example (6)

```
public class TCExample
{

   public void method () throws IOException,
                                NumberFormatException
   {
       BufferedReader br;
       String s;
       int num;

        System.out.print("Type in an integer: ");
        br = new BufferedReader(new
          InputStreamReader(System.in));
        s = br.readLine();
        num = Integer.parseInt(s);
    }
}
```

When The `Driver.Main ()` Method Can't Handle The Exception

```
public class Driver
{
    public static void main (String [] args) throws IOException,
      NumberFormatException
    {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

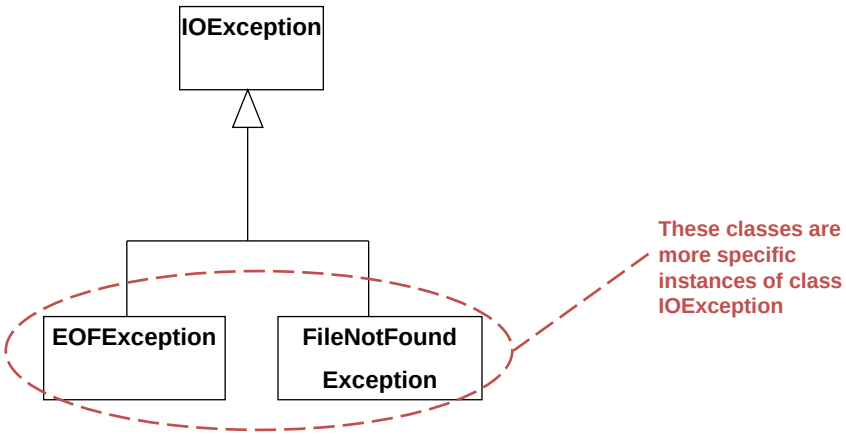
After This Section You Should Now Know

- The benefits of handling errors with an exception handler rather than employing a series of return values and conditional statements/branches.
- How to handle exceptions
  - Being able to call a method that may throw an exception by using a `try-catch` block
  - What to do if the caller cannot properly handle the exception
  - What is the finally clause, how does it work and when should it be used
- The effect of the inheritance hierarchy when catching exceptions

Simple File Input And Output

You will learn how to write to and read from text files in Java.

Inheritance Hierarchy For `IOExceptions`



Inheritance And Catching Exceptions

- If you are catching a sequence of exceptions then make sure that you catch the exceptions for the child classes before you catch the exceptions for the parent classes
- Deal with the more specific case before handling the more general case

Branches: Specific Before General

- **Correct**

```
If (x > 100)
   body;
else if (x > 10)
   body;
else if (x > 0)
   body;
```

- **Incorrect**

```
If (x > 0)
   body;
else if (x > 10)
   body;
else if (x > 100)
   body;
```

Inheritance And Catching Exceptions (2)

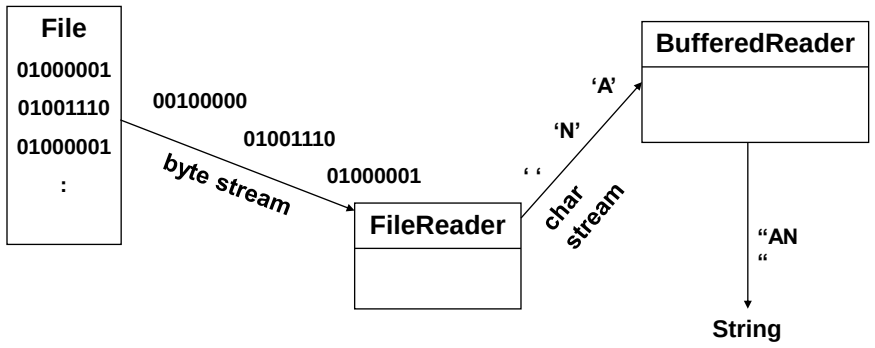
Correct

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

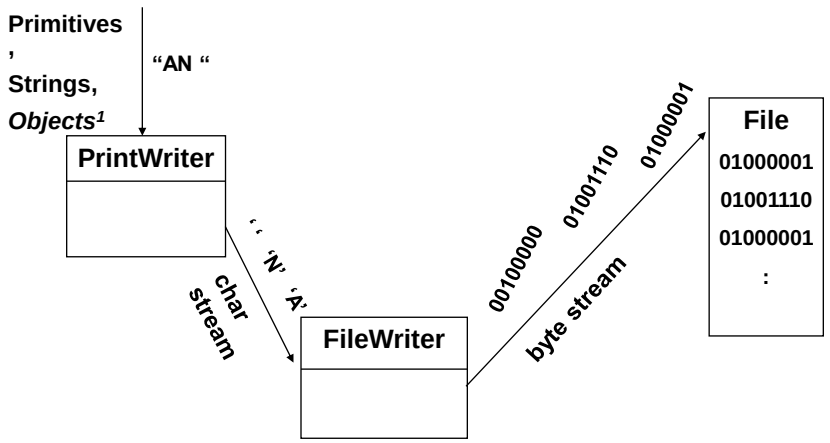
Incorrect

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```

Reading Text Input From A File



Writing Text Output To A File



1 By objects we of course mean references to objects

File Input And Output: One Complete Example

Location of the online example:
/home/219/examples/fileIO/Driver.java

```
public class Driver
{
    final static int MAX = 4;
    public static void main(String [] args)
    {
        String line = null;
        String [] paragraph = null;
        int i;
        Scanner in;

        // File IO
        PrintWriter pw = null;
        FileWriter fw = null;
        BufferedReader br = null;
        FileReader fr = null;

        in = new Scanner(System.in);
        paragraph = new String[MAX];
```

James Tam

File IO: Get Data And Write To File

```
// Get paragraph information from the user.
for (i = 0; i < MAX; i++)
{
    System.out.print("Enter line of text: ");
    line = in.nextLine();
    paragraph[i] = line;  //Add line as array element
}

// Write paragraph to file
try
{
    fw = new FileWriter("data.txt"); // Open
    pw = new PrintWriter(fw);
    for (i = 0; i < MAX; i++)
        pw.println(paragraph[i]);
    fw.close();                      // Close
}
catch (IOException e)
{
    System.out.println("Error writing to file");
}
```

James Tam

File IO: Read Data From File

```
try {
    fr = new FileReader("data.txt");  // Open
    br = new BufferedReader(fr);
    line = br.readLine();

    if (line == null)
        System.out.println("Empty file, nothing to read");

    while (line != null) {
        System.out.println(line);
        line = br.readLine();
    }
    fr.close();                       // Close
 }
 catch (FileNotFoundException e) {
     System.out.println("Could not open data.txt");
 }
 catch (IOException e) {
     System.out.println("Trouble reading from data.txt");
 }
```

James Tam

You Should Now Know

- How to write to files with Java classes
  - `FileWriter`
  - `PrintWriter`
- How to reading text information from files with Java classes
  - `FileReader`
  - `BufferedReader`