Java Exception Handling

Handling errors using Java's exception handling mechanism

James Tam

Approaches For Dealing With Error Conditions

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

James Tam

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);
        }
```

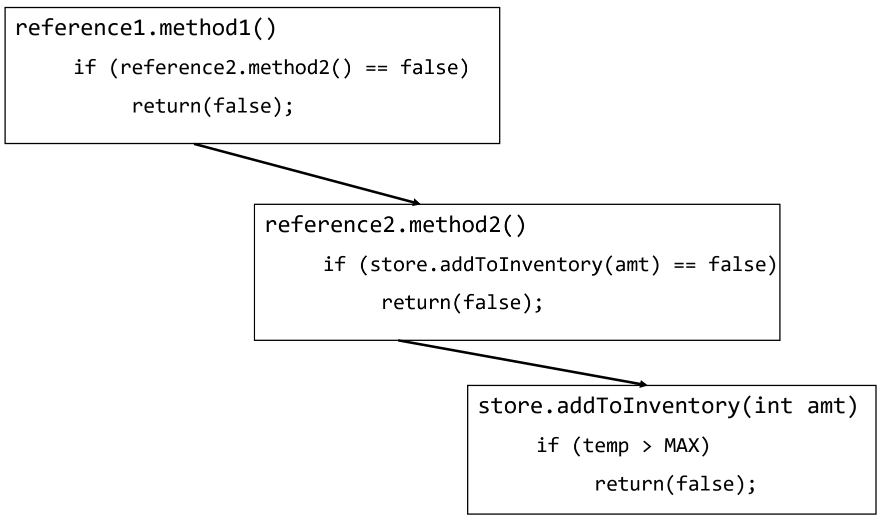
James Tam

Class Inventory: An Earlier Example (2)

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

James Tam

Some Hypothetical Method Calls: Condition/Return



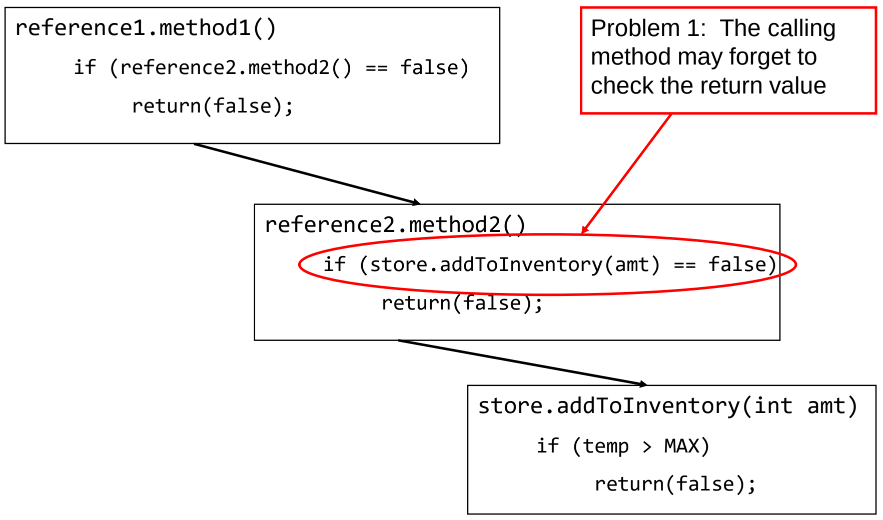
```
reference1.method1()
     if (reference2.method2() == false)
          return(false);
```

```
reference2.method2()
     if (store.addToInventory(amt) == false)
          return(false);
```

```
store.addToInventory(int amt)
     if (temp > MAX)
          return(false);
```

James Tam

Some Hypothetical Method Calls: Condition/Return



```
reference1.method1()
     if (reference2.method2() == false)
          return(false);
```

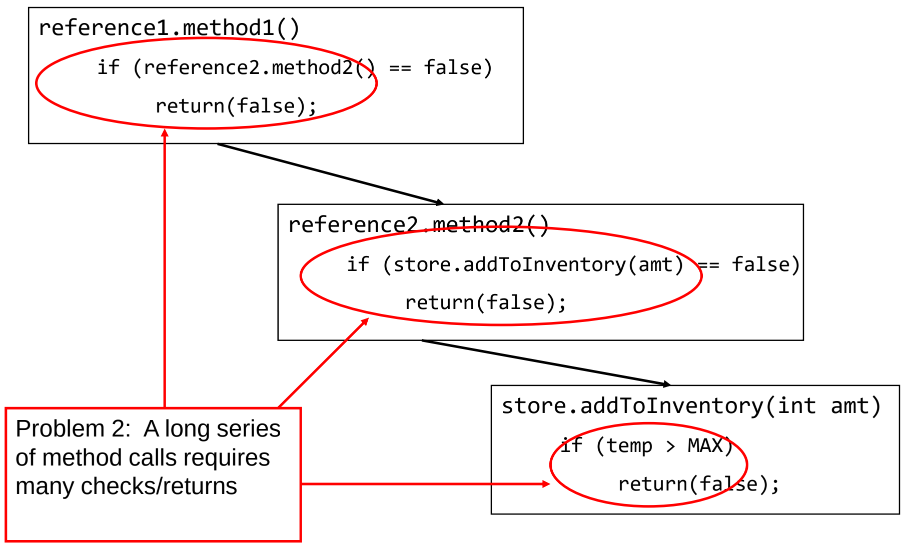
Problem 1: The calling method may forget to check the return value

```
reference2.method2()
     if (store.addToInventory(amt) == false)
          return(false);
```

```
store.addToInventory(int amt)
     if (temp > MAX)
          return(false);
```

James Tam

Some Hypothetical Method Calls: Condition/Return



```
reference1.method1()
        if (reference2.method2() == false)
                return(false);
```

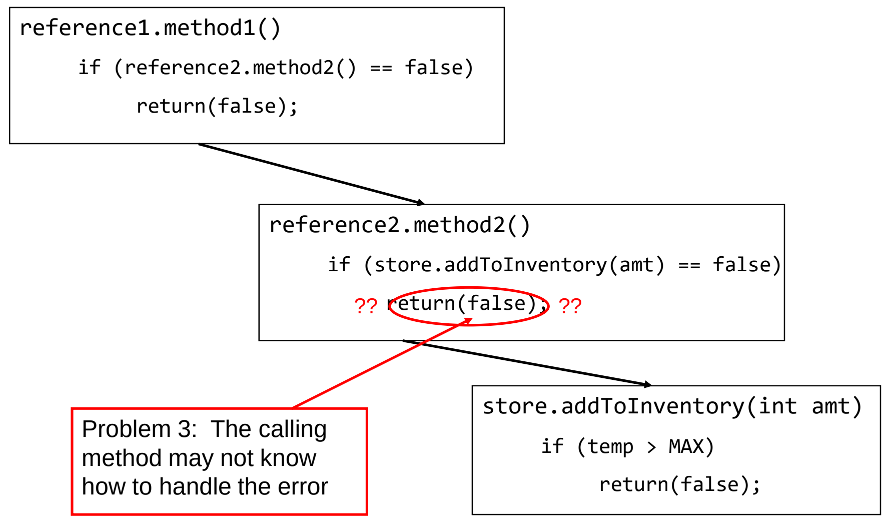
```
reference2.method2()
        if (store.addToInventory(amt) == false)
                return(false);
```

```
store.addToInventory(int amt)
        if (temp > MAX)
                return(false);
```

Problem 2: A long series of method calls requires many checks/returns

James Tam

Some Hypothetical Method Calls: Condition/Return



```
reference1.method1()
        if (reference2.method2() == false)
                return(false);
```

```
reference2.method2()
        if (store.addToInventory(amt) == false)
            ?? return(false); ??
```

```
store.addToInventory(int amt)
        if (temp > MAX)
                return(false);
```

Problem 3: The calling method may not know how to handle the error

James Tam

Approaches For Dealing With Error Conditions

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

James Tam

Handling Exceptions

Format:

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

James Tam

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);
```

James Tam

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)
        {
            ...
        }
    }
}
```

James Tam

Handling Exceptions: Where The Exceptions Occur

The first exception can occur here

```
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);
 }
```

James Tam

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);
 }
```

James Tam

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;
     ...
}
```

James Tam

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

James Tam

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;
     ...
}
```

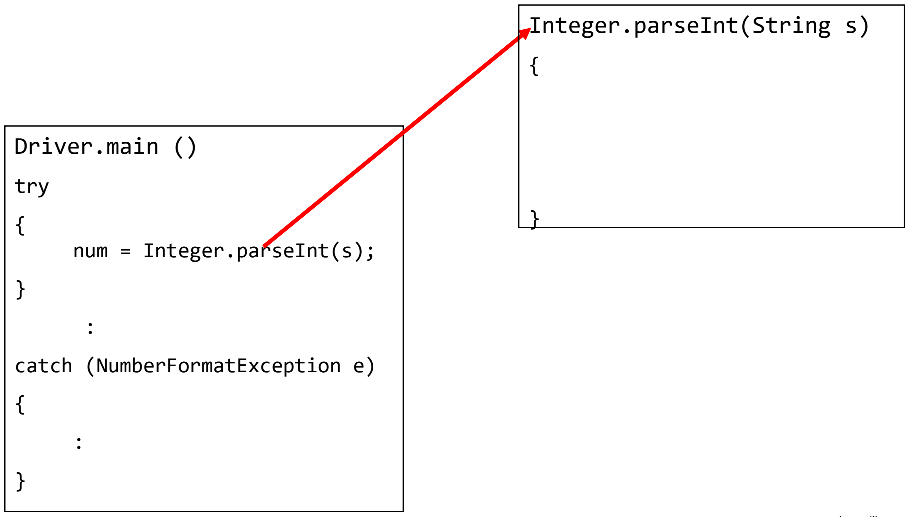
James Tam

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)
         {
              ...
        }
    }
}
```

James Tam

Handling Exceptions: Tracing The Example

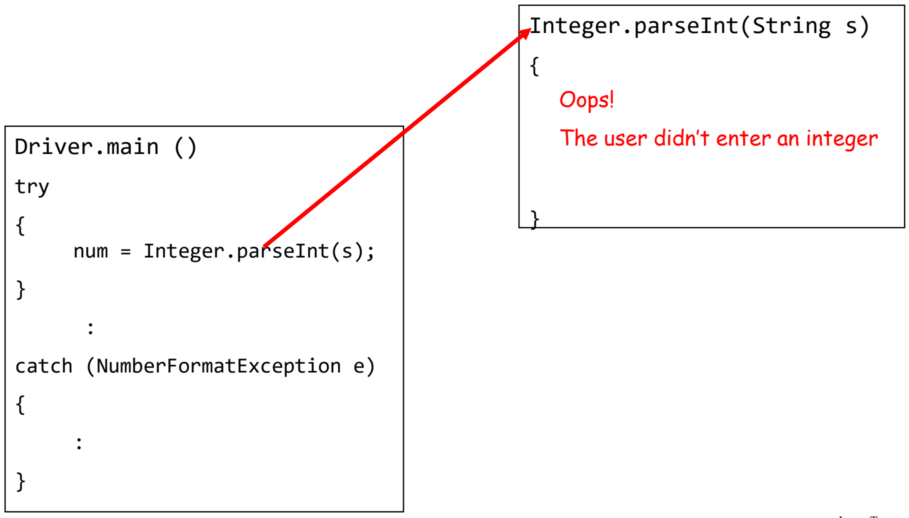


```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
     :
}
```

```
Integer.parseInt(String s)
{


}
```

James Tam

Handling Exceptions: Tracing The Example

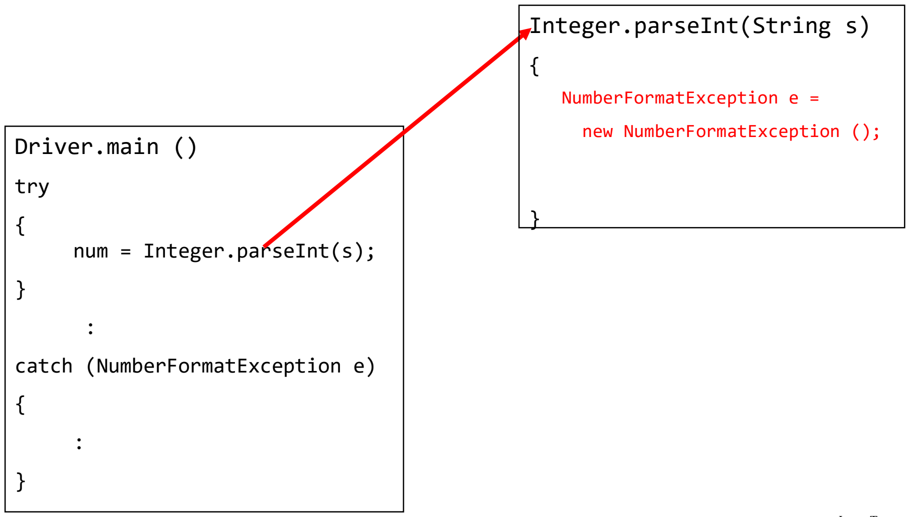


```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
     :
}
```

```
Integer.parseInt(String s)
{
   Oops!
   The user didn't enter an integer

}
```

James Tam

Handling Exceptions: Tracing The Example

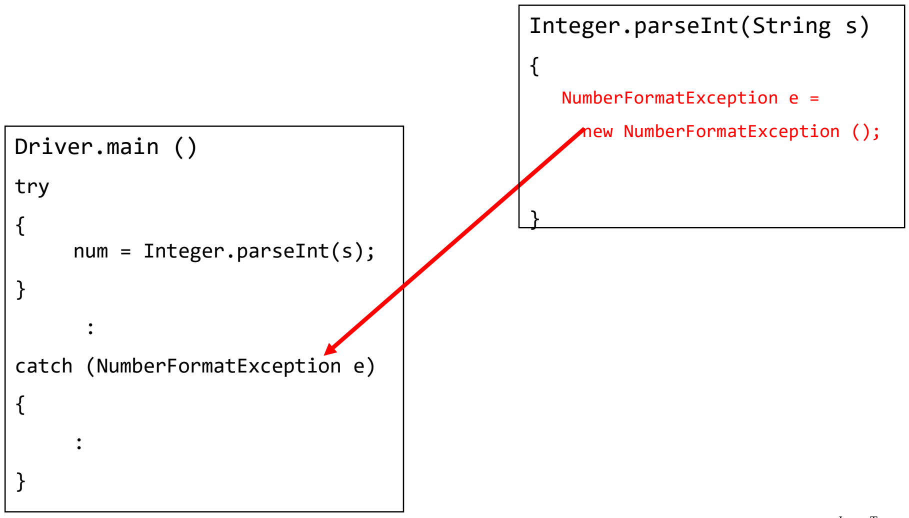


```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
     :
}
```

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

}
```

James Tam

Handling Exceptions: Tracing The Example



```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
     :
}
```

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

}
```

James Tam

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();
        }
     }
}
```

James Tam

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)
```

James Tam

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
}
```

James Tam

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
}
```

NO!

James Tam

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.

James Tam

The Finally Clause: Exception Thrown

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

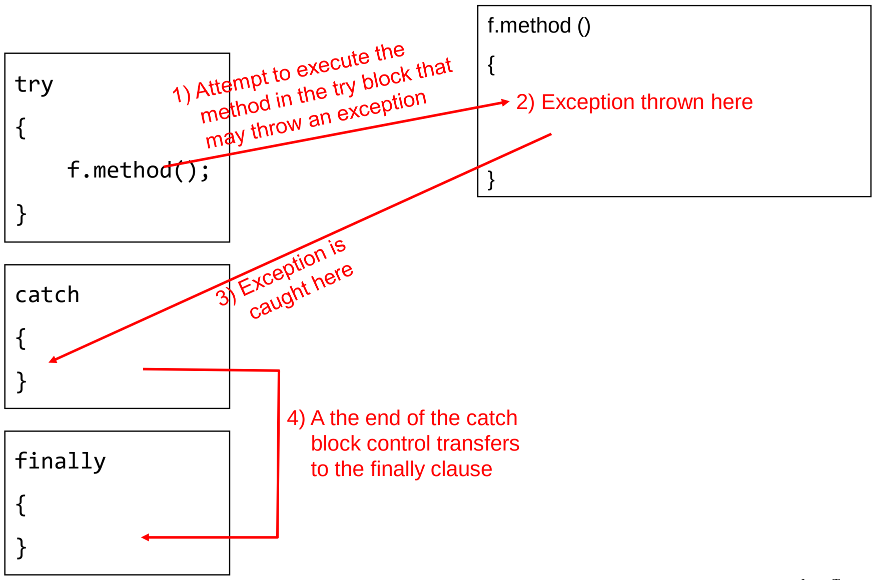
```
catch
{
}
```

```
finally
{
}
```

```
f.method ()
{


}
```

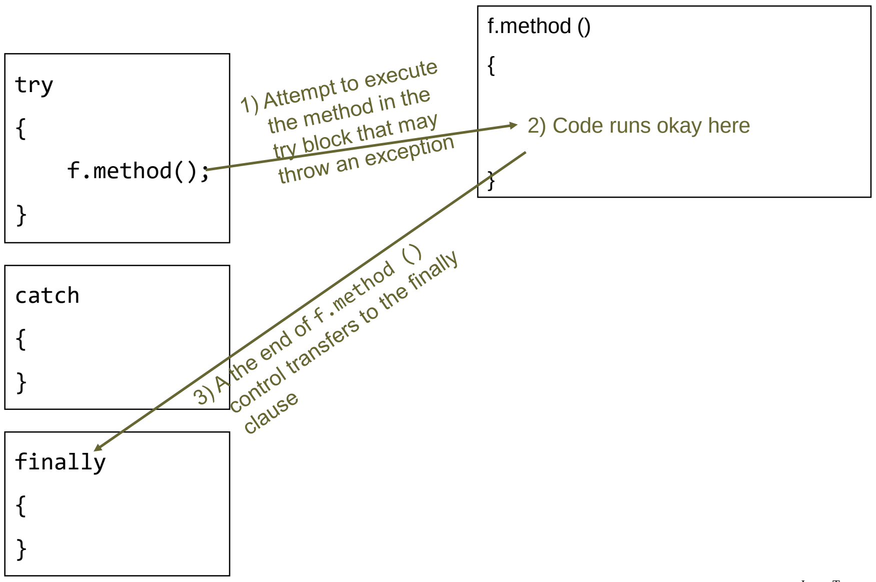
James Tam

The Finally Clause: Exception Thrown


try
{
f.method();
}
catch
{
}
finally
{
}
f.method ()
{
}
1) Attempt to execute the method in the try block that may throw an exception
2) Exception thrown here
3) Exception is caught here
4) A the end of the catch block control transfers to the finally clause

James Tam

The Finally Clause: No Exception Thrown


try
{
f.method();
}
catch
{
}
finally
{
}
f.method ()
{
}
1) Attempt to execute the method in the try block that may throw an exception
2) Code runs okay here
3) A the end of f.method () control transfers to the finally clause

James Tam

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();
    }
}
```

James Tam

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;
         }
```

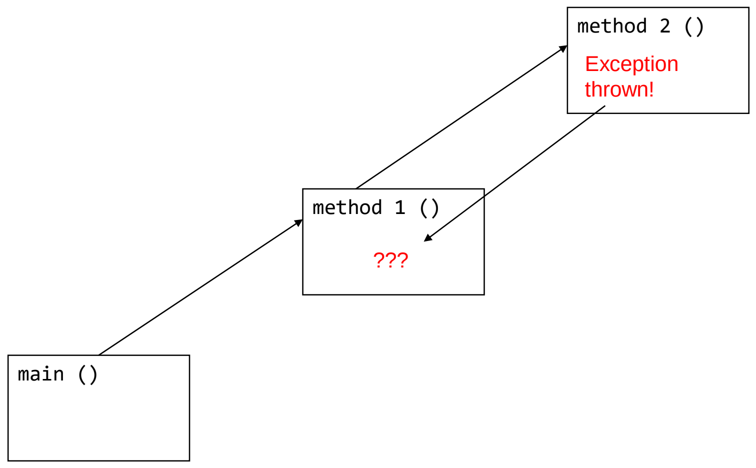
James Tam

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;
            }
        }
    }
```

James Tam

When The Caller Can't Handle The Exceptions



James Tam

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

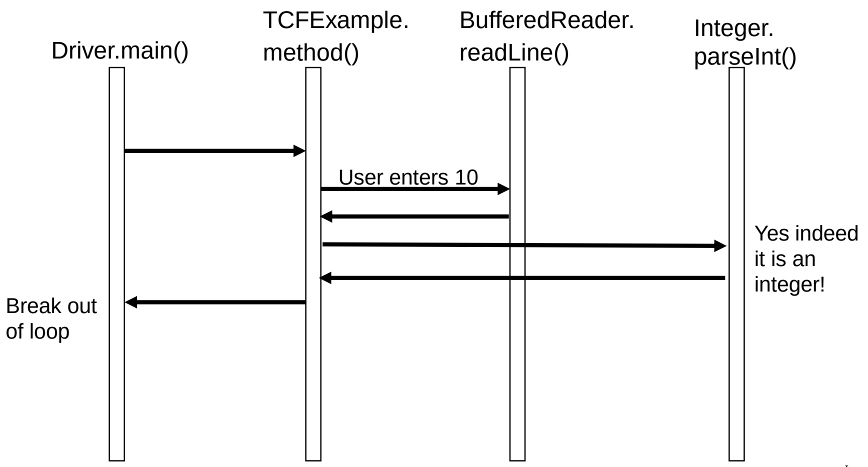
Location of the online example:

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

James Tam

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

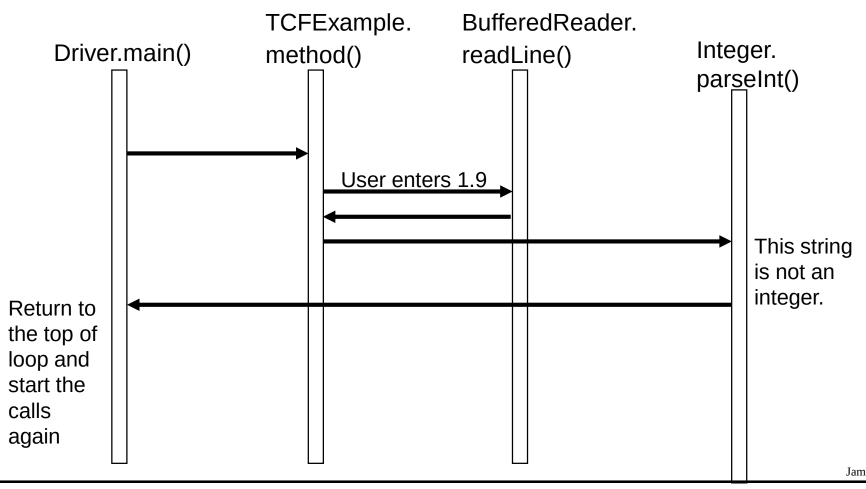
- Tracing the method calls when *no exception occurs*:



James Tam

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

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



James Tam

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;
```

James Tam

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
```

James Tam

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);
    }
}
```

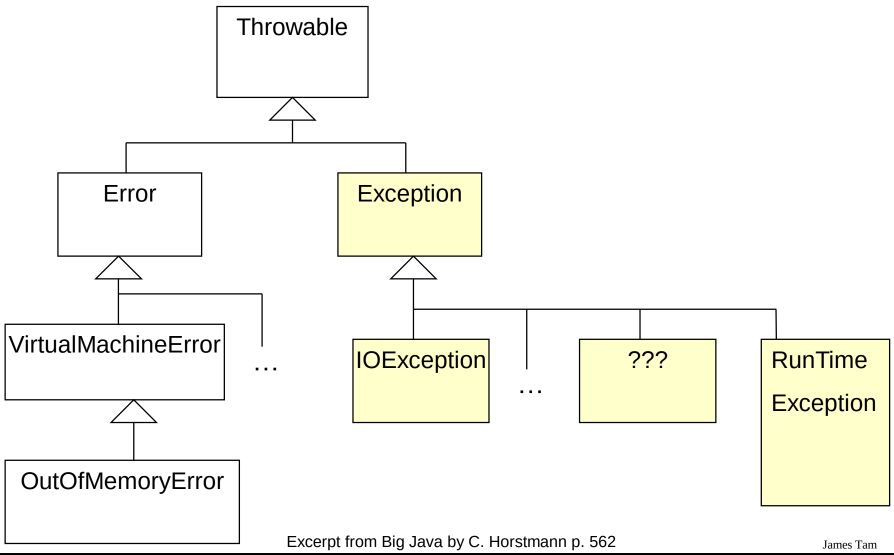
James Tam

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();
    }
}
```

James Tam

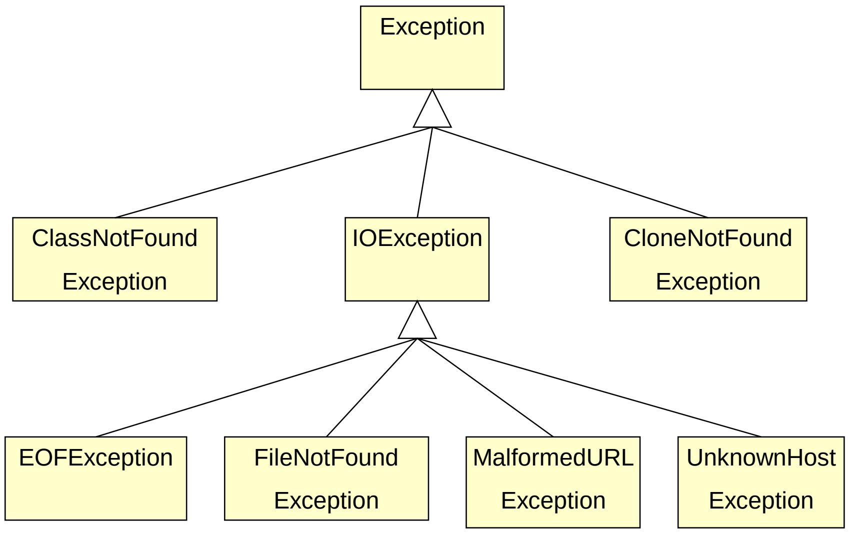
Creating Your Own Exceptions (If There Is Time)



Excerpt from Big Java by C. Horstmann p. 562

James Tam

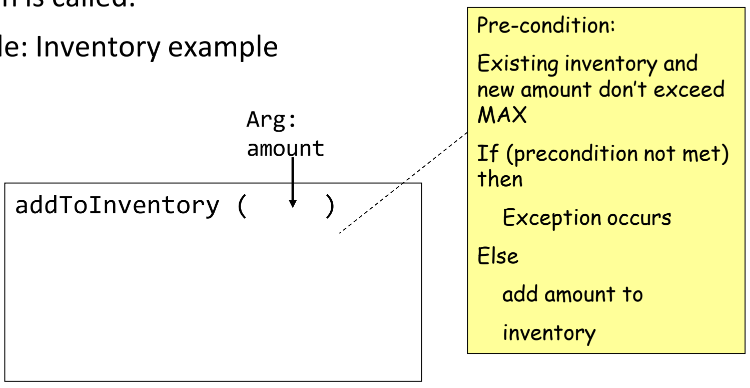
Class Exception: The Local Inheritance Hierarchy



James Tam

Writing New Exceptions

- Typical approach: tie the exception into preconditions
- Remember: preconditions are things that must be true when a function is called.
- Example: Inventory example

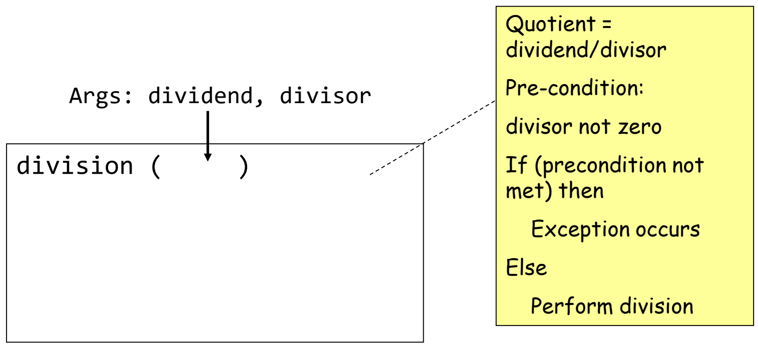


```
Pre-condition:
Existing inventory and new amount don't exceed MAX
If (precondition not met) then
    Exception occurs
Else
    add amount to
    inventory
```

James Tam

Writing New Exceptions (2)

- Example 2: Division



```
Quotient = dividend/divisor
Pre-condition:
divisor not zero
If (precondition not met) then
    Exception occurs
Else
    Perform division
```

Writing New Exceptions: An Example

Location of the online example:

`/home/219/examples/exceptions/writingExceptions/inventoryExample`

Writing New Exceptions: Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Inventory chinook = new Inventory();
        try
        {
           chinook.add(10);
        }
        catch (InventoryOverMaxException e)
        {
           System.out.print(">>Too much to be added to
                             stock<<");
        }
```

James Tam

Writing New Exceptions: Driver Class (2)

```
        System.out.println(chinook.showStockLevel ());
        try
        {
           chinook.add(10);
        }
        catch (InventoryOverMaxException e)
        {
           System.out.println(">>Too much to be added to
                               stock<<");
        }
```

James Tam

Writing New Exceptions: `Driver` Class (3)

```
    System.out.println(chinook.showStockLevel ());
    try
    {
       chinook.add(100);
    }
    catch (InventoryOverMaxException e)
    {
       System.out.println(">>Too much to be added to
                         stock<<");
    }
```

James Tam

Writing New Exceptions: `Driver` Class (4)

```
        System.out.println(chinook.showStockLevel ());
        try
        {
           chinook.remove(21);
        }
        catch (InventoryUnderMinException e)
        {
           System.out.println(">>Too much to remove from
           stock<<");
        }
        System.out.println(chinook.showStockLevel ());
    }
}
```

James Tam

Writing New Exceptions: Class Inventory

```
public class Inventory
{
    public final int CRITICAL = 10;
    public final int MIN = 0;
    public final int MAX = 100;
    private int stockLevel = 0;

     public boolean inventoryTooLow()
     {
         if (stockLevel < CRITICAL)
             return true;
         else
             return false;
    }
```

James Tam

Writing New Exceptions: Class Inventory (2)

```
 public void add(int amount) throws InventoryOverMaxException
    {
       int temp;
       temp = stockLevel + amount;
       if (temp > MAX)
       {
          throw new InventoryOverMaxException("Adding " +
             amount + " item(s) “ +
             "will cause stock to become greater than "
             + MAX + " units");
      }
     else
           stockLevel = stockLevel + amount;
     }
```

"Throws":

- An exception of type <E> can occur in this method

"Throw":

- Instantiates an exception of type <E>
- Execution transfers back to the '`catch`' block of the caller

James Tam

Writing New Exceptions: Class Inventory (3)

```
  public void remove(int amount) throws
    InventoryUnderMinException {
      int temp;
      temp = stockLevel - amount;
      if (temp < MIN) {
          throw new InventoryUnderMinException("Removing " +
            amount + " item(s) will cause stock to become less
 "
            than " + MIN + " units");
      }
      else
          stockLevel = temp;
    }

    public String showStockLevel ()  {
        return("Inventory: " + stockLevel);
    }
}
```

James Tam

Writing New Exceptions: Class InventoryOverMaxException

```
public class InventoryOverMaxException extends Exception
{
    public InventoryOverMaxException()
    {
        super ();
    }

    public InventoryOverMaxException(String s)
    {
        super (s);
    }
}
```

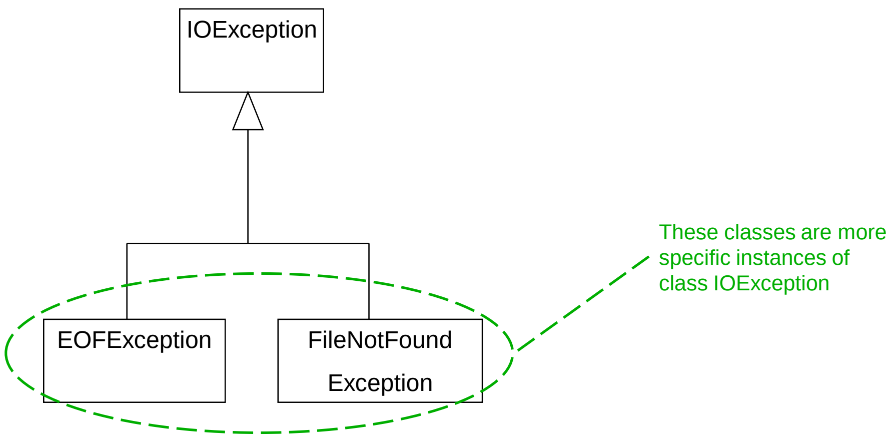
James Tam

Writing New Exceptions: Class InventoryUnderMinException

```
public class InventoryUnderMinException extends Exception
{
    public InventoryUnderMinException()
    {
        super();
    }

    public InventoryUnderMinException(String s)
    {
        super(s);
    }
}
```

James Tam

Inheritance Hierarchy For IOExceptions


IOException
EOFException
FileNotFound Exception
These classes are more specific instances of class IOException

James Tam

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

James Tam

Inheritance And Catching Exceptions (2)

Correct

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

Incorrect

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```

James Tam

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
- How to write your classes of exceptions
- The effect of the inheritance hierarchy when catching exceptions

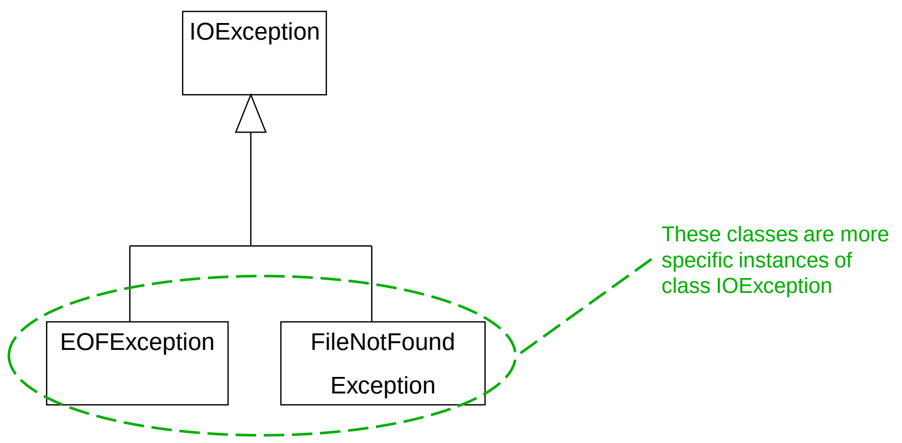
James Tam

Simple File Input And Output

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

James Tam

Inheritance Hierarchy For `IOExceptions`


IOException
EOFException
FileNotFound Exception
These classes are more specific instances of class IOException

James Tam

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

James Tam

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;
```

James Tam

Inheritance And Catching Exceptions (2)

Correct

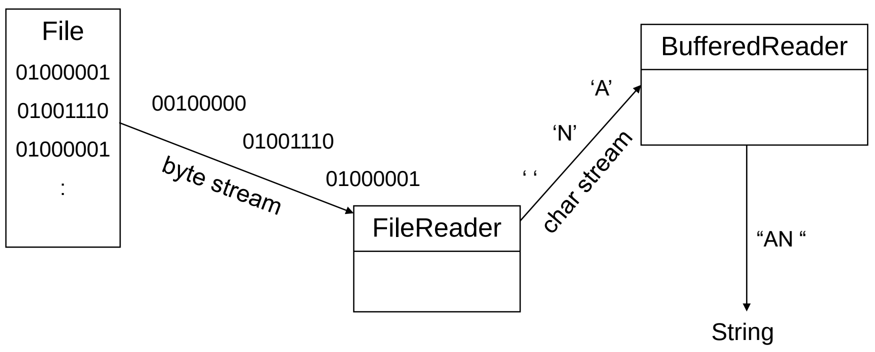
```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

Incorrect

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```

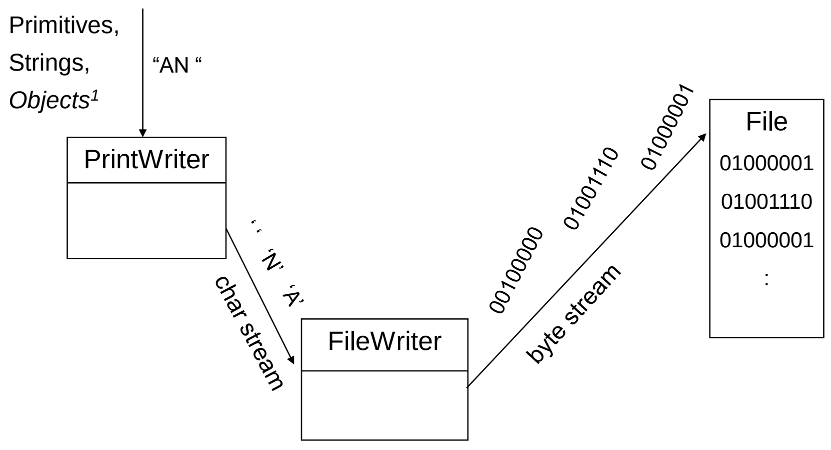
James Tam

Reading Text Input From A File



James Tam

Writing Text Output To A File



1 By objects we of course mean references to objects

James Tam

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`

James Tam