

# 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 addToInventory (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;
        }
    }
}
```

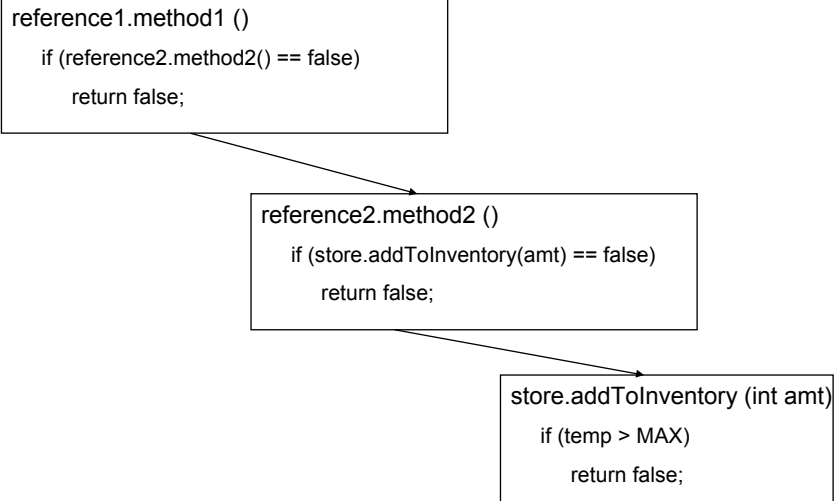
James Tam

## Class Inventory: An Earlier Example (2)

```
else
{
    stockLevel = stockLevel + amount;
    return true;
}
} // End of method addToInventory
:
```

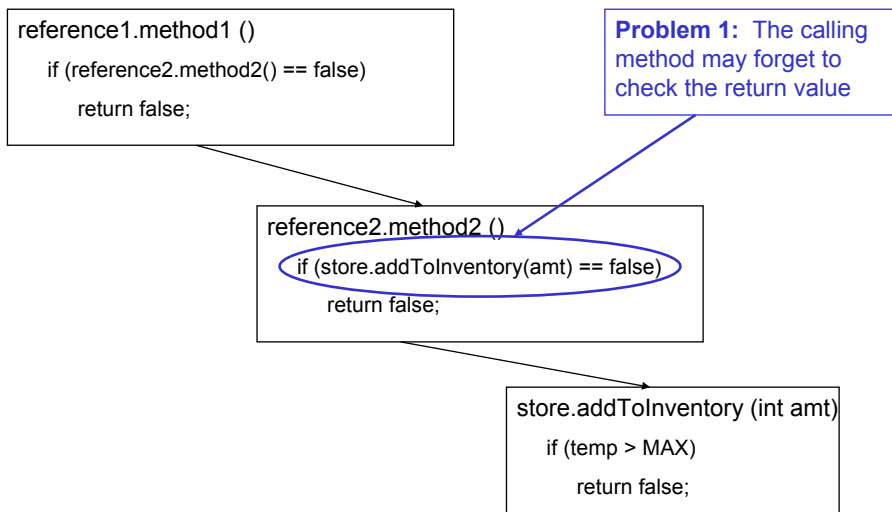
James Tam

## Some Hypothetical Method Calls: Condition/Return



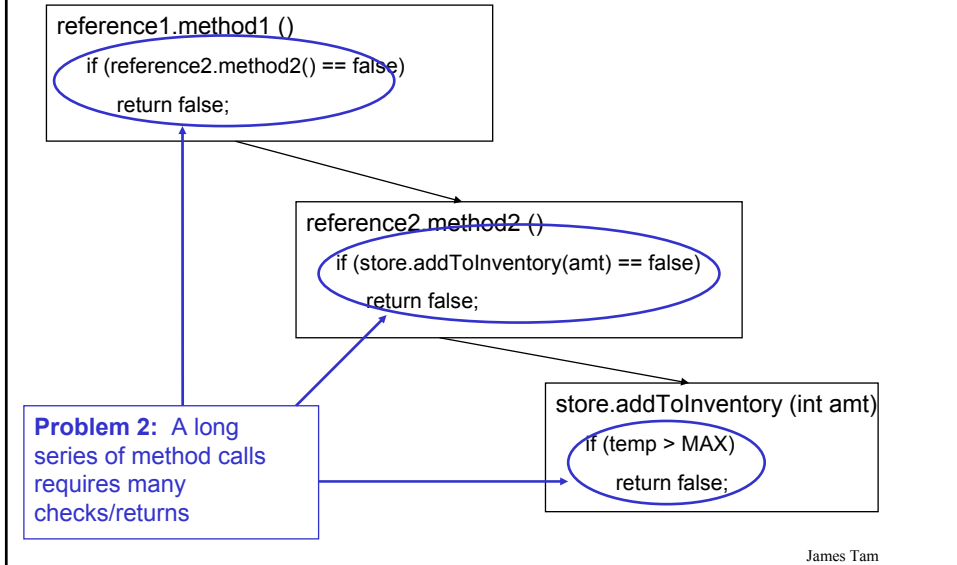
James Tam

## Some Hypothetical Method Calls: Condition/Return

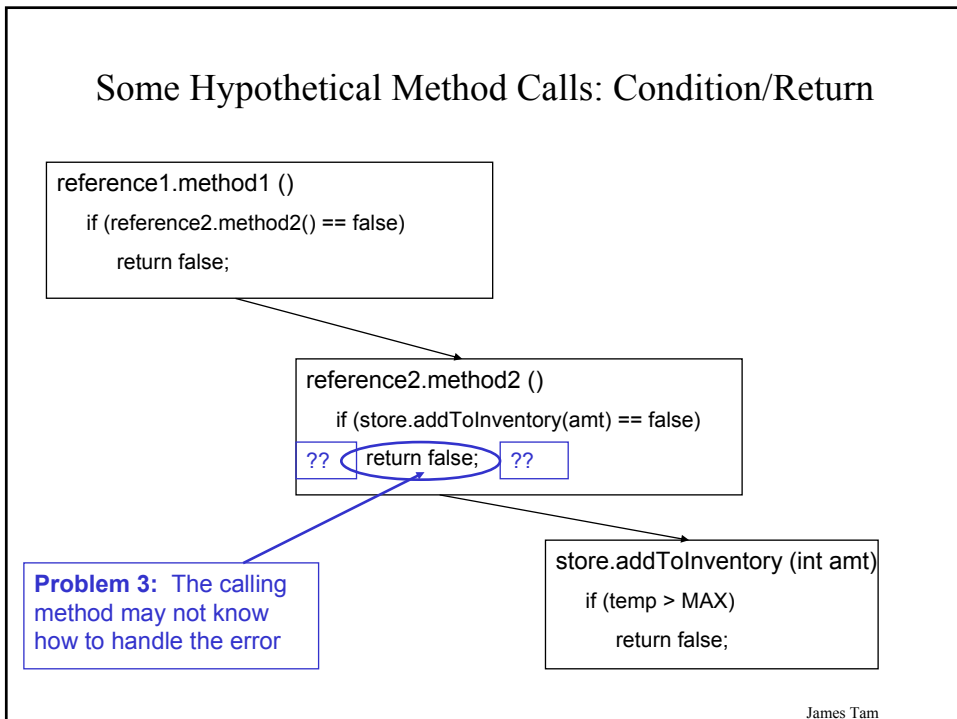


James Tam

## 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

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

The complete program can be found in the directory:  
/home/233/examples/exceptions/handlingExceptions/inputExample

```
import java.io.*;

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

```
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 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);
}
```

The first exception can occur here

James Tam

## Where The Exceptions Occur In Class BufferedReader

For online documentation for this class go to:

<http://java.sun.com/j2se/1.5.0/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 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://java.sun.com/j2se/1.5.0/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

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

```
Integer.parseInt (String s)
{
}
}
```

James Tam

## Handling Exceptions: Catching The Exception

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

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();
}
}
```

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

## Categories Of Exceptions

- Unchecked exceptions
- Checked exception

James Tam

## Characteristics Of Unchecked Exceptions

- The compiler doesn't require you to catch them if they are thrown.
  - *No try-catch block required by the compiler*
- They can occur at any time in the program (not just for a specific method)
- Typically they are fatal runtime errors that are beyond the programmer's control
  - Use branches/decision making statements rather than the exception handling model.
- Examples:
  - `NullPointerException`, `IndexOutOfBoundsException`, `ArithmeticException`...

James Tam

## Common Unchecked Exceptions: NullPointerException

```
int [] arr = null;  
arr[0] = 1;
```

← **NullPointerException**

```
arr = new int [4];  
int i;  
for (i = 0; i <= 4; i++)  
    arr[i] = i;  
  
arr[i-1] = arr[i-1] / 0;
```

James Tam

## Common Unchecked Exceptions: ArrayIndexOutOfBoundsException

```
int [] arr = null;  
arr[0] = 1;  
  
arr = new int [4];  
int i;  
for (i = 0; i <= 4; i++)  
    arr[i] = i;
```

← **ArrayIndexOutOfBoundsException**  
(when i = 4)

```
arr[i-1] = arr[i-1] / 0;
```

James Tam


## Common Unchecked Exceptions: ArithmeticExceptions

```
int [] arr = null;  
arr[0] = 1;
```

```
arr = new int [4];  
int i;  
for (i = 0; i <= 4; i++)  
    arr[i] = i;
```

```
arr[i-1] = arr[i-1] / 0;
```

**ArithmeticException**  
(Division by zero)



James Tam

## Checked Exceptions

Must be handled if the potential for an error exists

- You must use a try-catch block

Deal with problems that occur in a specific place

- When a particular method is invoked you must enclose it within a try-catch block

Example:

- NumberFormatException, IOException

James Tam

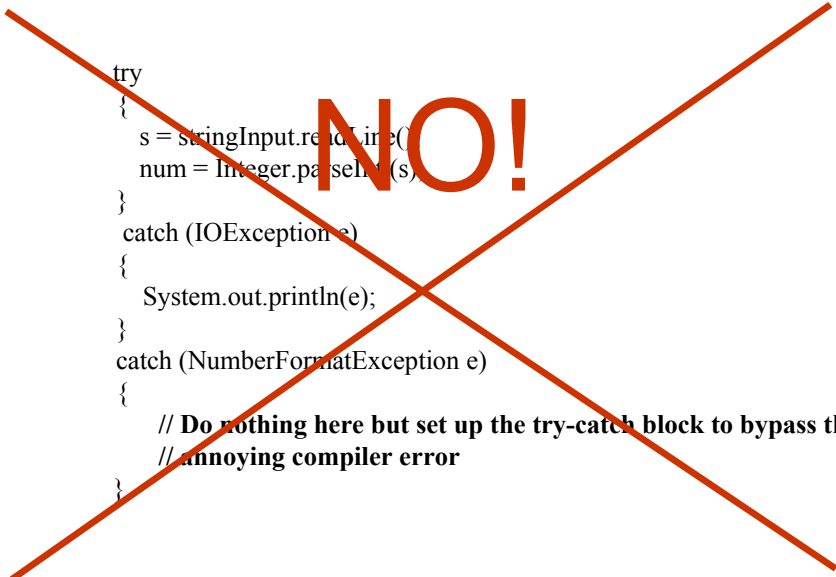


## Avoid Squelching Your Exceptions

```
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
}
```

James Tam

## 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");
}
```

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();  
}
```

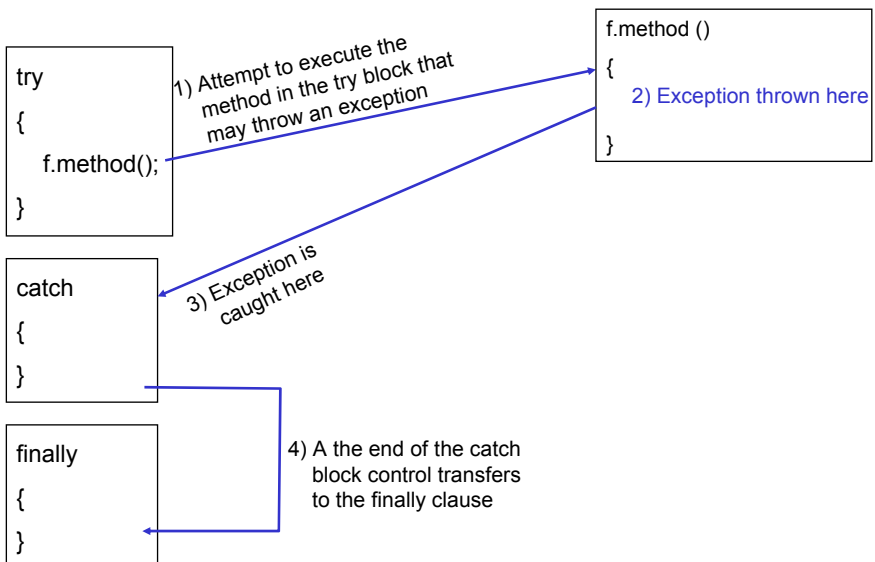
```
catch  
{  
}
```

```
finally  
{  
}
```

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

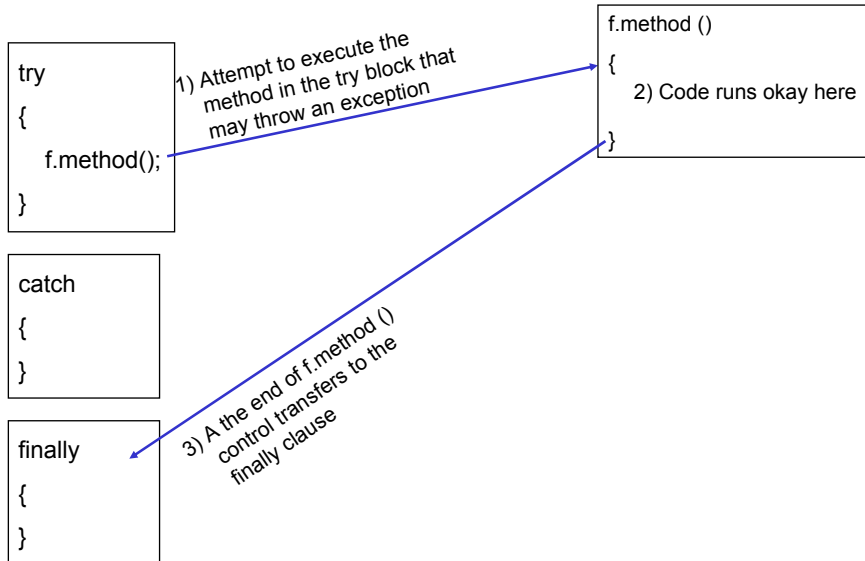
James Tam

## The Finally Clause: Exception Thrown



James Tam

## The Finally Clause: No Exception Thrown



## Try-Catch-Finally: An Example

The complete program can be found in the directory:  
/home/233/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;
        }
    }
}
```

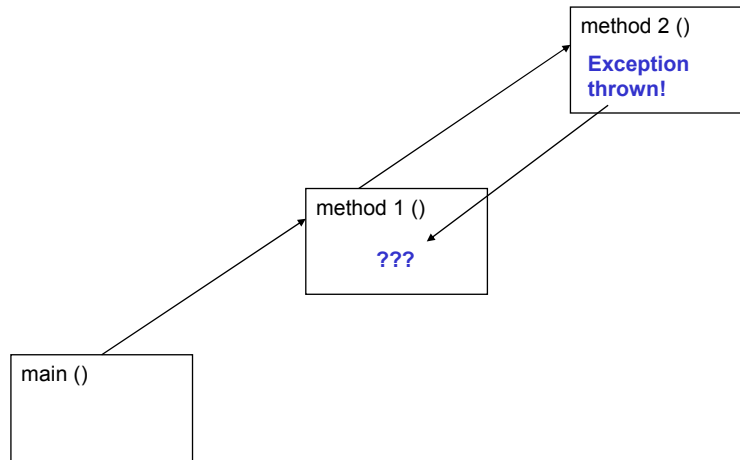
James Tam

## 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

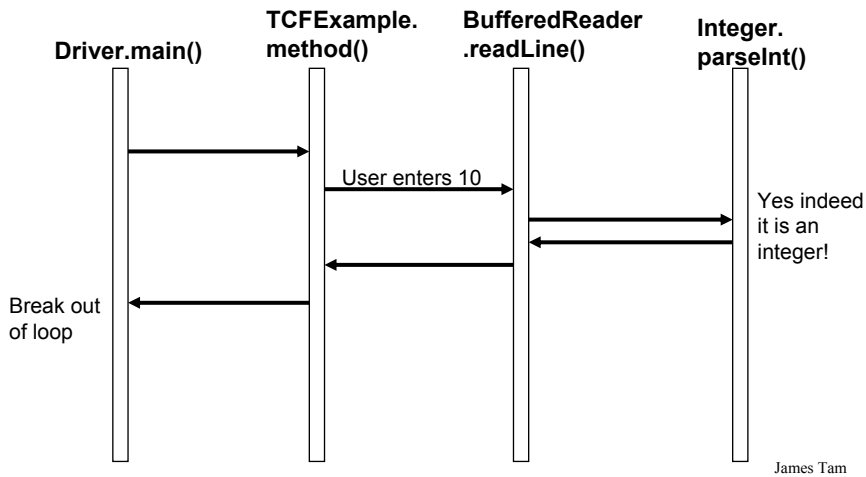
## When The Caller Can't Handle The Exceptions: An Example

The complete program can be found in the directory:  
`/home/233/examples/exceptions/handlingExceptions/  
delegatingExceptions`

James Tam

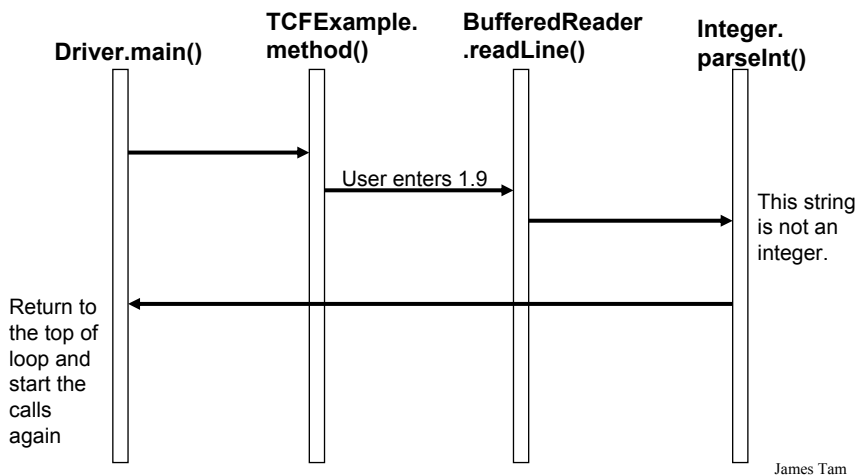
## When The Caller Can't Handle The Exceptions: An Example (2)

Tracing the method calls when *no exception occurs*:



## 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;
```

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)

```
import java.io.*;

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);
    }
}
```

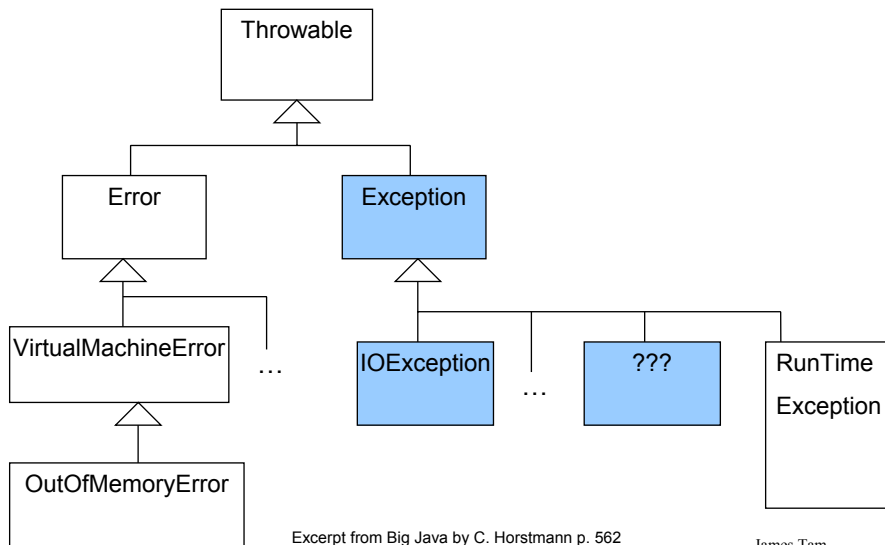
James Tam

## When The Main () Method Can't Handle The Exception

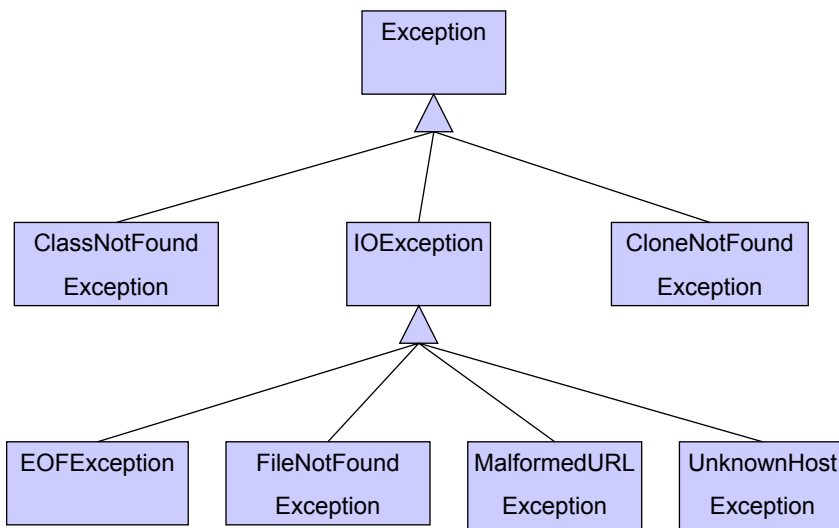
```
class Driver
{
    public static void main (String [] args) throws IOException,
        NumberFormatException
    {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

James Tam

## Creating Your Own Exceptions



## Class Exception: The Local Inheritance Hierarchy



## Writing New Exceptions: An Example

The full example can be found in the directory:  
`/home/233/examples/exceptions/writingExceptions/inventoryExample`

James Tam

## Writing New Exceptions: The Driver Class

```
public class Driver
{
    public static void main (String [] args)
    {
        CommandProcessor userInterface = new CommandProcessor ();
        userInterface.processSelection();
    }
}
```

James Tam

## Writing New Exceptions: Class CommandProcessor

```
public class CommandProcessor
{
    private char menuSelection;
    private Inventory chinook = new Inventory();

    public void display ()
    {
        System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
        System.out.println("\t(a)dd new stock to inventory");
        System.out.println("\t(c)heck if stock level is critically low");
        System.out.println("\t(d)isplay stock level");
        System.out.println("\t(q)uit program");
        System.out.println("\t(r)emove stock from inventory");
        System.out.print("Selection: ");
    }
}
```

James Tam

## Writing New Exceptions: Class CommandProcessor (2)

```
public void getSelection ()
{
    menuSelection = (char) Console.in.readChar();
    Console.in.readLine();
}

public void processSelection ()
{
    int amount;
    boolean amountInvalid = false;
    do
    {
        display();
        getSelection ();
    }
}
```

James Tam

## Writing New Exceptions: Class CommandProcessor (3)

```
switch (menuSelection)
{
    case 'A':
    case 'a':
        do
        {
            System.out.print("No. items to add: ");
            amount = Console.in.readInt();
            Console.in.readLine();
            try
            {
                chinook.add(amount);
                amountInvalid = false;
            }
        }
    }
}
```

James Tam

## Writing New Exceptions: Class CommandProcessor (4)

```
catch (InventoryOverMaxException anException)
{
    amountInvalid = true;
    System.out.println(anException);
    System.out.println
        ("Enter a value equal to or " + "less than " +
         (Inventory.MAX - chinook.getStockLevel()));
}
finally
{
    System.out.println(chinook.showStockLevel());
}
} while (amountInvalid == true);
break; // End of 'add' case
```

James Tam

## Writing New Exceptions: Class CommandProcessor (5)

```
case 'C':
case 'c':
    if (chinook.inventoryTooLow())
        System.out.println("Stock levels critical!");
    else
        System.out.println("Stock levels okay");
    System.out.println(chinook.showStockLevel());
    break;

case 'D':
case 'd':
    System.out.println(chinook.showStockLevel());
    break;

case 'Q':
case 'q':
    System.out.println("Quitting program");
    break;
```

James Tam

## Writing New Exceptions: Class CommandProcessor (6)

```
case 'R':
case 'r':
    do
    {
        System.out.print("No. items to remove: ");
        amount = Console.in.readInt();
        Console.in.readLine();
        try
        {
            chinook.remove(amount);
            amountInvalid = false;
        }
        catch (InventoryUnderMinException anException)
        {
            amountInvalid = true;
            System.out.println(anException);
            System.out.println("Enter a value equal to or "
                + "less than " + chinook.getStockLevel());
        }
    }
```

James Tam

## Writing New Exceptions: Class CommandProcessor (7)

```
        finally
        {
            System.out.println(chinook.showStockLevel());
        }
    } while (amountInvalid == true);
    break;

    default:
        System.out.println("Enter one of 'a', 'c', 'd', 'q' or 'r'");
    }
} while ((menuSelection != 'Q') && (menuSelection != 'q'));
} // End of switch
} // End of method
```

James Tam

## Class Inventory

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

    public Inventory ()
    {
        stockLevel = 0;
    }
    public boolean inventoryTooLow ()
    {
        if (stockLevel < CRITICAL)
            return true;
        else
            return false;
    }
}
```

James Tam

## Class Inventory

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

    public Inventory ()
    {
        stockLevel = 0;
    }
    public boolean inventoryTooLow ()
    {
        if (stockLevel < CRITICAL)
            return true;
        else
            return false;
    }
}
```

James Tam

## Class Inventory (2)

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

James Tam



## Class Inventory (3)

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

James Tam

## Class Inventory (4)

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

public int getStockLevel ()
{
    return stockLevel;
}
} // End of class Inventory
```

James Tam

## Class InventoryOverMaxException

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

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

James Tam

## Class InventoryUnderMinException

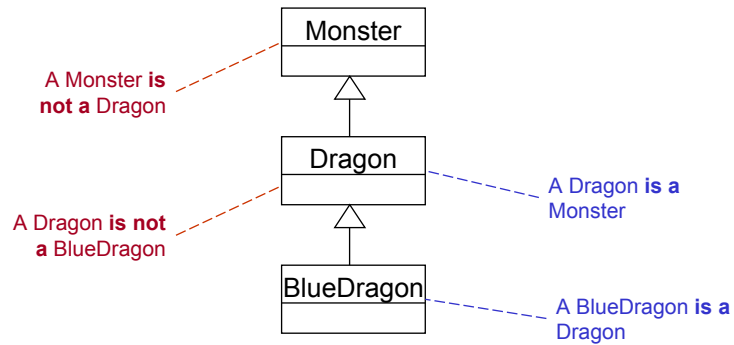
```
public class InventoryUnderMinException extends Exception
{
    public InventoryUnderMinException ()
    {
        super();
    }

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

James Tam

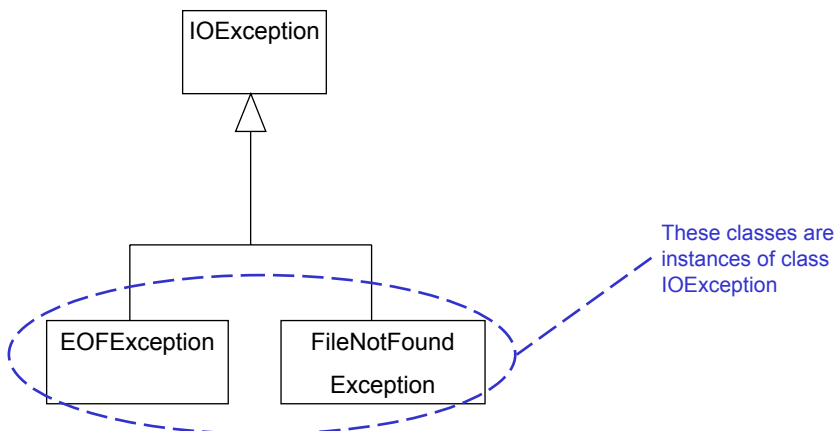
## Review: Inheritance

- Remember: You can substitute instances of a sub class for instances of a super class.
- This is because the sub class **is an** instance of the super class



James Tam

## Inheritance Hierarchy For IOExceptions



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

## 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
- What is the difference between a checked and an unchecked exception
- How to write your classes of exceptions
- The effect of the inheritance hierarchy when catching exceptions