Java Exception Handling

Handling errors using Java's exception handling mechanism

Approaches For Dealing With Error Conditions

- Use conditional statements and return values
- Use Java's exception handling mechanism

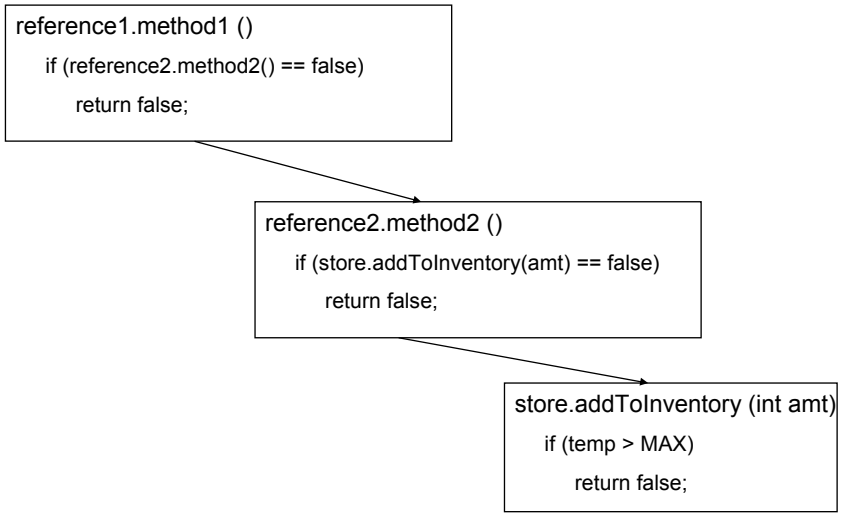
Class Inventory: An Earlier Example

```
public class Inventory
{
   public final int MIN = 0;
   public  final int MAX = 100;
   public final int CRITICAL = 10;
   public boolean 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;
      }
```


Class Inventory: An Earlier Example (2)

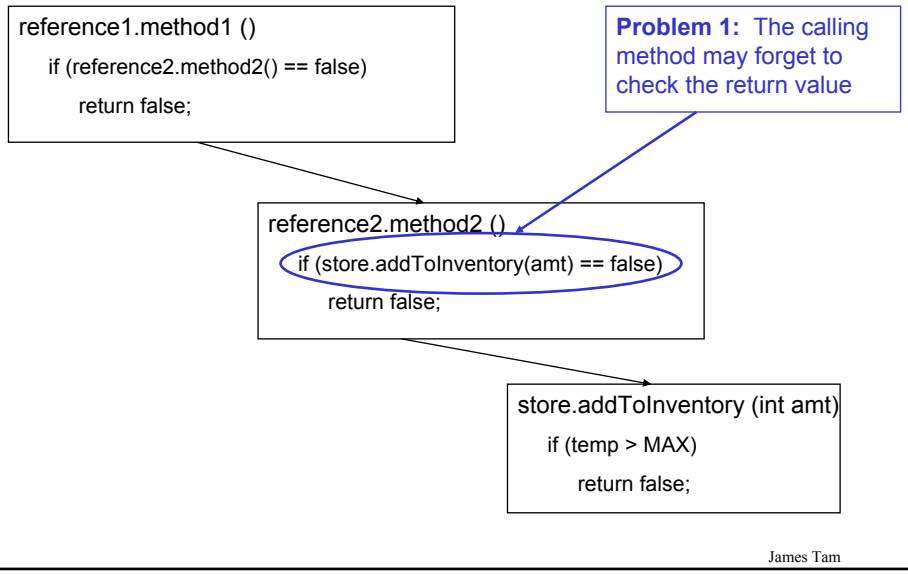
```
      else
      {
         stockLevel = stockLevel + amount;
         return true;
      }
   } // End of method addToInventory
   :
```

Some Hypothetical Method Calls: Condition/Return



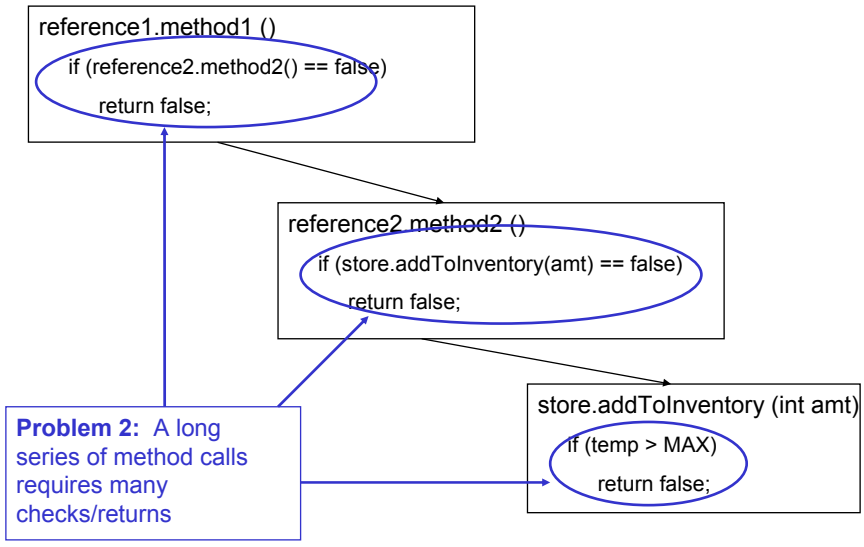
James Tam

Some Hypothetical Method Calls: Condition/Return



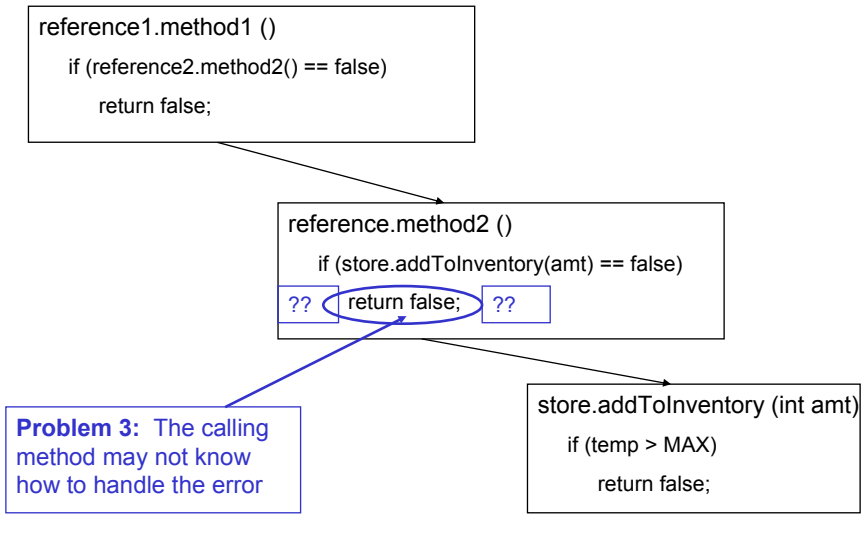
James Tam

Some Hypothetical Method Calls: Condition/Return



James Tam

Some Hypothetical Method Calls: Condition/Return



James Tam

Approaches For Dealing With Error Conditions

- Use conditional statements 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

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

```
import java.io.*;

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 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

Where The Exceptions Occur In Class BufferedReader

For online documentation for this class go to:
http://java.sun.com/j2se/1.4.1/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 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

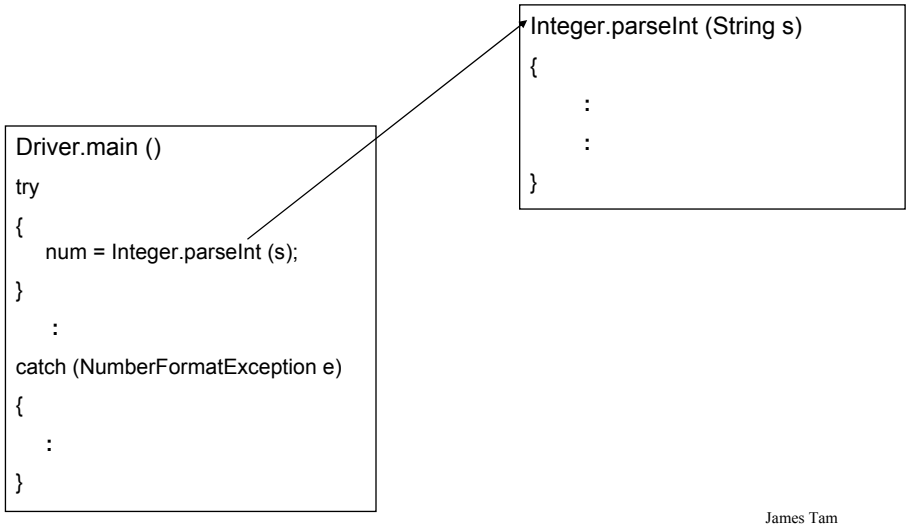
For online documentation for this class go to:
http://java.sun.com/j2se/1.4.1/docs/api/java/lang/Integer.html

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

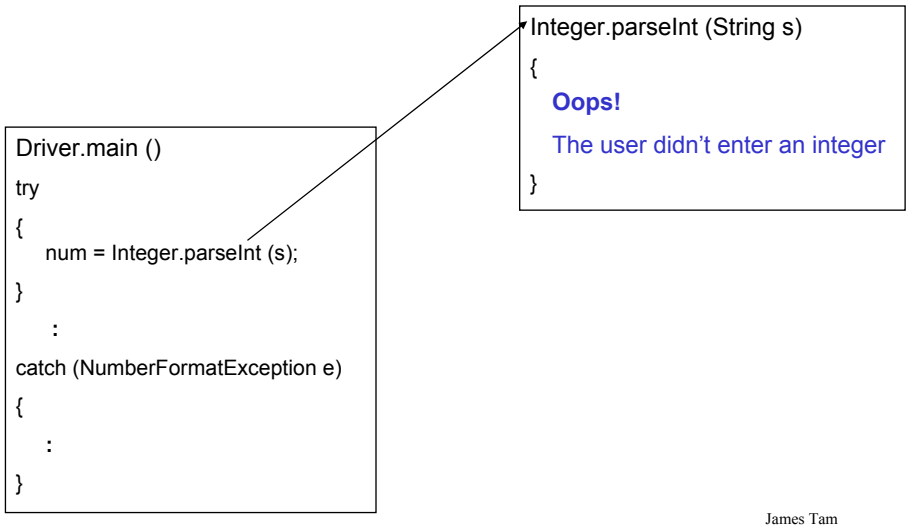

Handling Exceptions: The Details

```
      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)
      {
            :          :          :
      }
   }
}
```

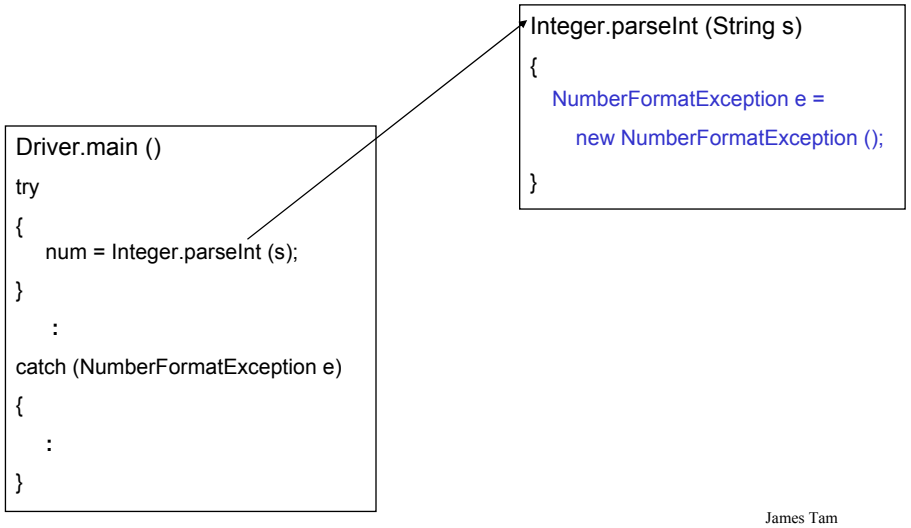
Handling Exceptions: Tracing The Example



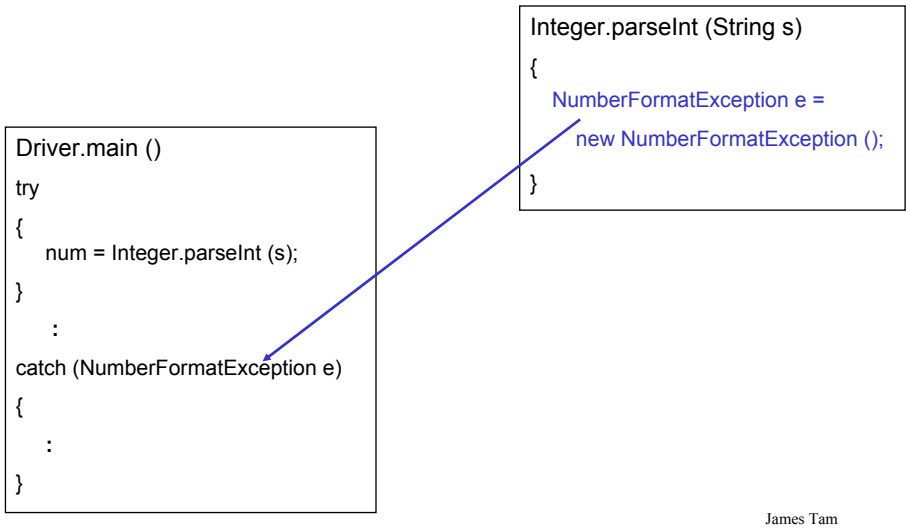
Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example



Handling Exceptions: Tracing The Example

```
Integer.parseInt (String s)
{



}
```

```
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(e.getMessage());
           System.out.println(e);
           e.printStackTrace();
        }
     }
}
```


Catching The Exception: Error Messages

```
        catch (NumberFormatException e)
        {
           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)

Categories Of Exceptions

- Unchecked exceptions
- Checked exception

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 conditional statements rather than the exception handling model.
- Examples:
  - NullPointerException,IndexOutOfBoundsException, ArithmeticException…

Common Unchecked Exceptions: NullPointerException

```
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;
```

NullPointerException

Common Unchecked Exceptions: ArrayIndexOutOfBoundsException

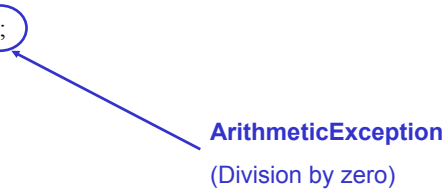
```
int [] arr = null;
arr[0] = 1;

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

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

ArrayIndexOutOfBoundsException

(when i = 4)

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;
```



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

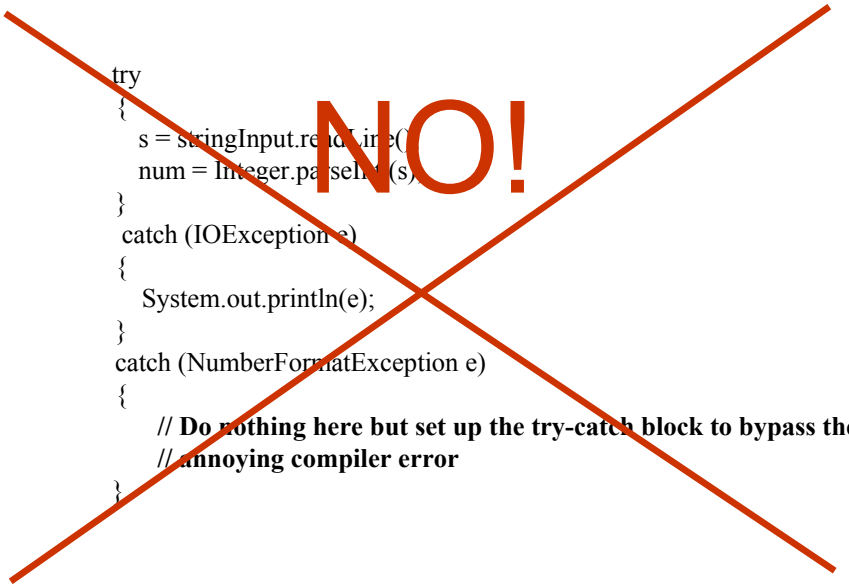
Example:

- NumberFormatException, IOException

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
}
```

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

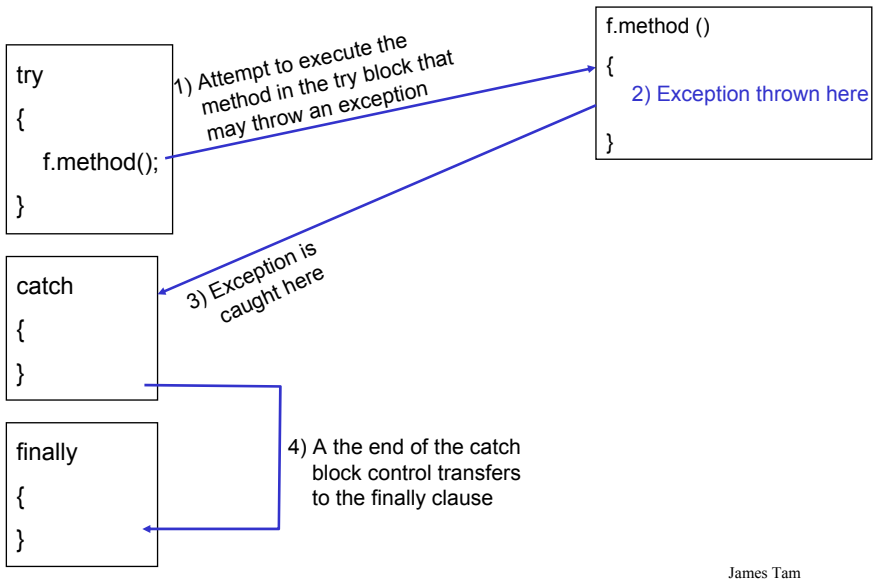
```
try
{
   f.method();
}
```

```
catch
{
}
```

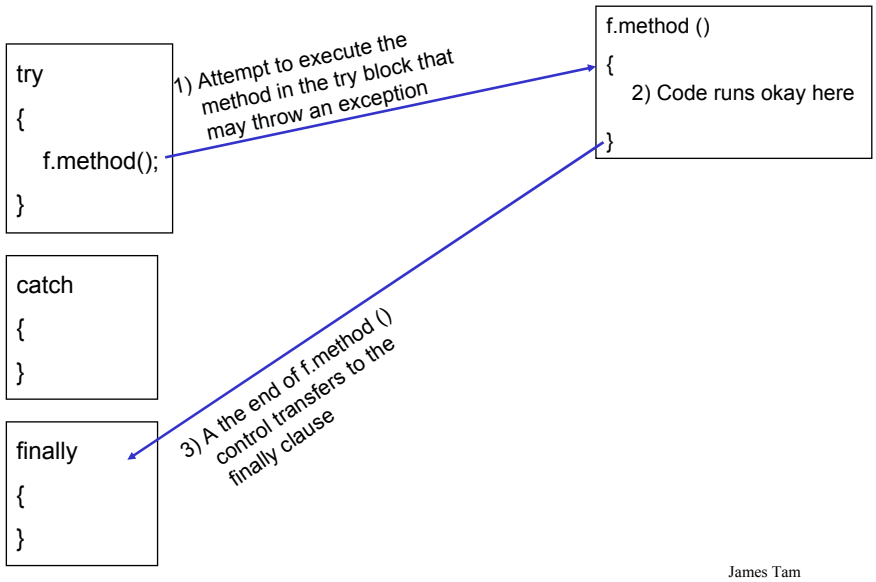
```
finally
{
}
```

```
Foo.method ()
{


}
```

The Finally Clause: Exception Thrown



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

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

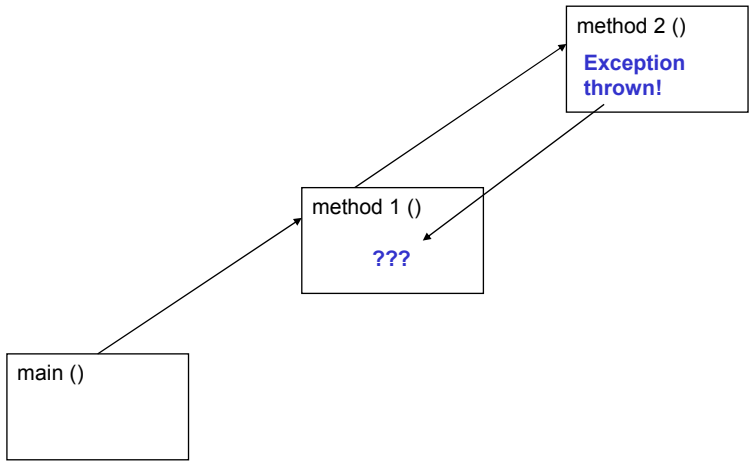

Try-Catch-Finally: An Example (2)

```
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

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

```
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 (2)

```
      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
```

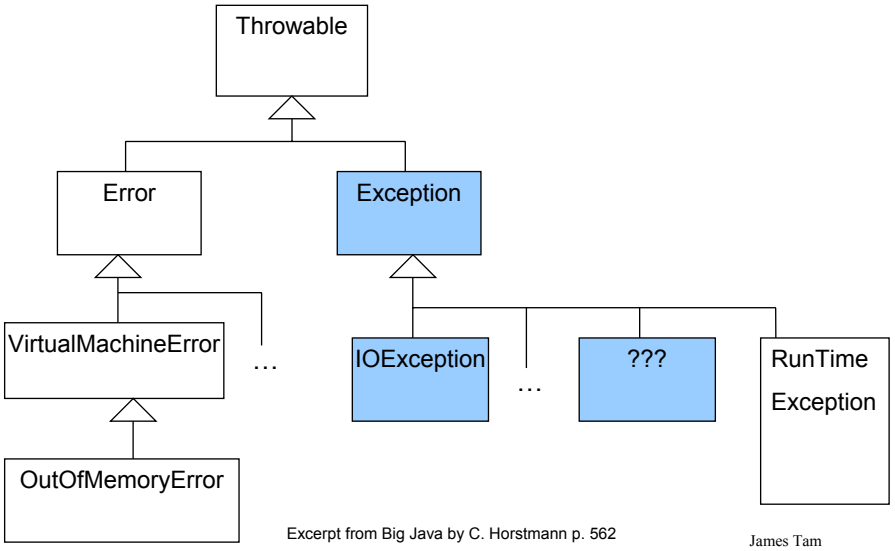
When The Caller Can't Handle The Exceptions: An Example (3)

```
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);
   }
}
```


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();
   }
}
```

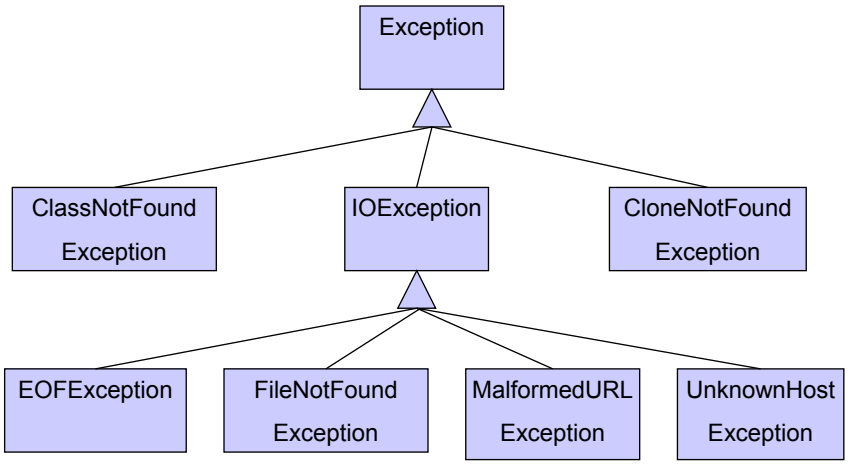
Creating Your Own Exceptions



Excerpt from Big Java by C. Horstmann p. 562

James Tam

Class Exception: The Local Inheritance Hierarchy



James Tam

Writing New Exceptions: An Example

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

Writing New Exceptions: The Driver Class

```
class Driver
{
   public static void main (String [] argv)
   {
      Inventory chinookInventory = new Inventory ();
      CommandProcessor userInterface = new CommandProcessor
            (chinookInventory);
      userInterface.startProcessingInput ();
   }
}
```

Writing New Exceptions: Class CommandProcessor

```
public class CommandProcessor
{
   private char menuOption;
   private Inventory storeInventory;

   public CommandProcessor (Inventory storeToTrack)
   {
      menuOption = 'q';
      storeInventory = storeToTrack;
   }
```


Writing New Exceptions: Class CommandProcessor (2)

```
   public void startProcessingInput ()
   {
      do
      {
         displayMenu();
         readMenuOption();
         switch (menuOption)
         {
            case 'a':
            case 'A':
               storeInventory.getAmountToAdd();
               break;

            case 'r':
            case 'R':
               storeInventory.getAmountToRemove();
               break;
```

Writing New Exceptions: Class CommandProcessor (3)

```
case 'd':
case 'D':
   storeInventory.displayInventoryLevel();
   break;

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

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


Writing New Exceptions: Class CommandProcessor (4)

```
        default:
           System.out.println("Enter one of A, R, D, C or Q");
        }
    } while ((menuOption != 'Q') && (menuOption != 'q'));
}  // End of method startProcessingInput
```

Writing New Exceptions: Class CommandProcessor (5)

```
    protected void displayMenu ()
    {
       System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
       System.out.println("\t(A)dd new stock to inventory");
       System.out.println("\t(R)emove stock from inventory");
       System.out.println("\t(D)isplay stock level");
       System.out.println("\t(C)heck if stock level is critical");
       System.out.print("\t(Q)uit program");
       System.out.println();
       System.out.print("Selection: ");
    }
    protected void readMenuOption ()
    {
       menuOption = (char) Console.in.readChar();
       Console.in.readLine();
       System.out.println();
    }
} // End of class CommandProcesor
```


The Inventory Class

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

   private int stockLevel;
   private boolean amountInvalid;
```

The Inventory Class (2)

```
public void getAmountToAdd ()
{
   int amount;
   do
   {
      System.out.print("No. items to add: ");
      amount = Console.in.readInt();
      Console.in.readLine();
      try
      {
         addToInventory(amount);
         amountInvalid = false;
      }
```


The Inventory Class (3)

```
      catch (InventoryOverMaxException e)
      {
         System.out.println(e);
         System.out.println("Enter another value.");
         System.out.println();
         amountInvalid = true;
      }
      finally
      {
         displayInventoryLevel();
      }
   } while (amountInvalid == true);
}  // End of method getAmountToAdd
```

The Inventory Class (4)

```
public void getAmountToRemove ()
{
   int amount;
   do
   {
      System.out.print("No. items to remove: ");
      amount = Console.in.readInt();
      Console.in.readLine();
      try
      {
         removeFromInventory(amount);
         amountInvalid = false;
      }
```


The Inventory Class (5)

```
      catch (InventoryBelowMinException e)
      {
         System.out.println(e);
         System.out.println("Enter another value.");
         System.out.println();
         amountInvalid = true;
      }
      finally
      {
         displayInventoryLevel();
      }
   } while (amountInvalid == true);
}  // End of method getAmountToRemove
```

The Inventory Class (6)

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


The Inventory Class (7)

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

The Inventory Class (8)

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

    public void displayInventoryLevel ()
    {
       System.out.println("No. items in stock: " + stockLevel);
    }

} // End of class Inventory
```


Class InventoryOverMaxException

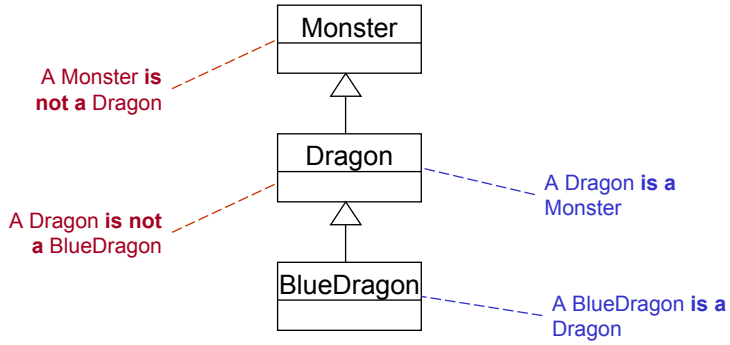
```
public class InventoryOverMaxException extends Exception
{
   public InventoryOverMaxException ()
   {
      super ();
   }

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

Class InventoryBelowMinException

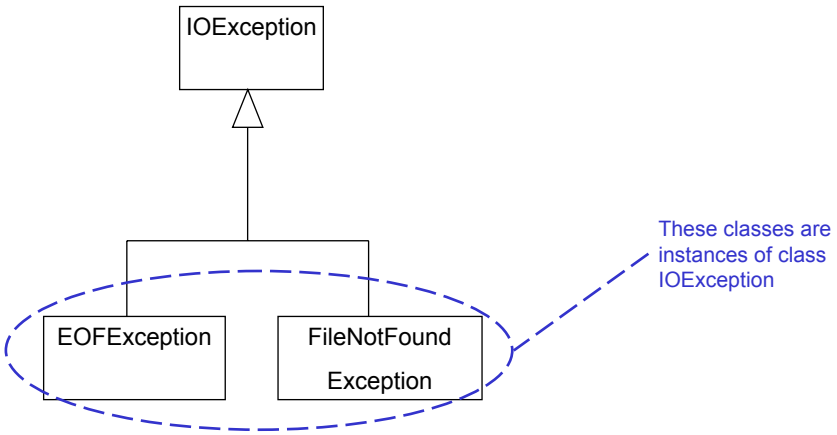
```
public class InventoryBelowMinException extends Exception
{
   public InventoryBelowMinException ()
   {
      super();
   }

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


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



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

Inheritance And Catching Exceptions (2)

Correct

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

Incorrect

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```


You Should Now Know

- The benefits of handling errors with an exception handler rather than employing a series of return values and conditional statements.
- 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