# 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

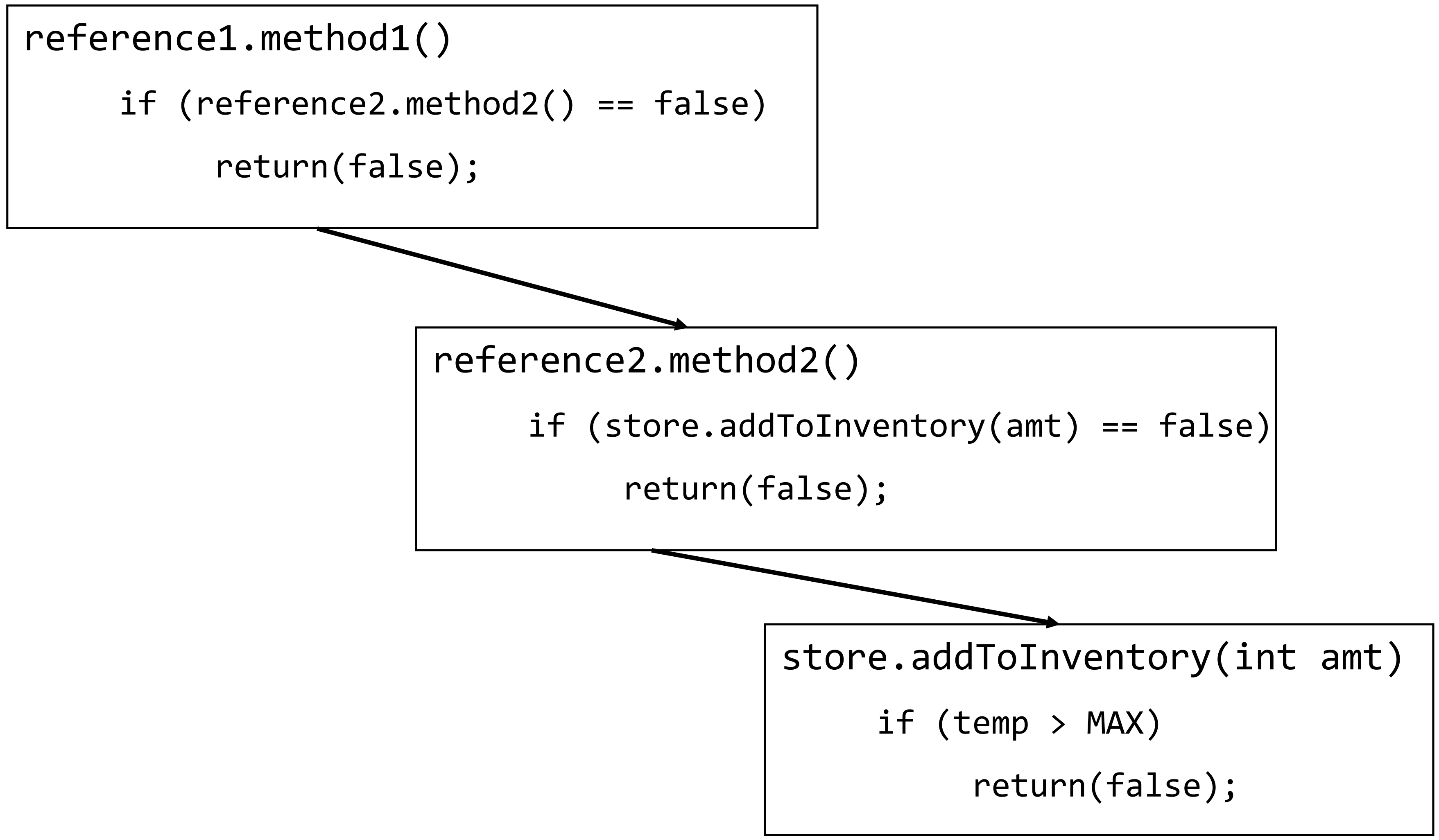
## Class Inventory: An Earlier Example

```
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

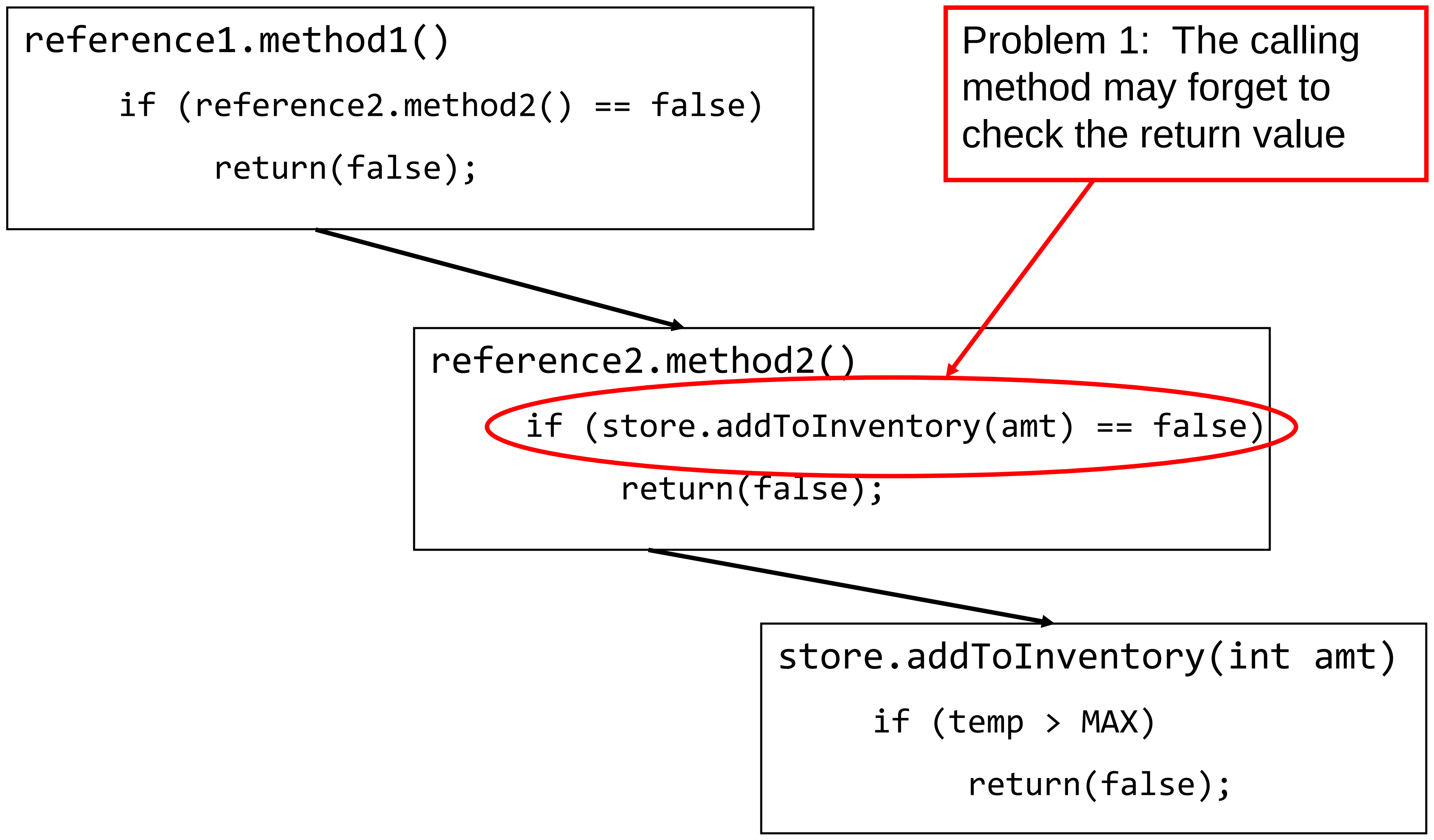


```
reference1.method1()
     if (reference2.method2() == false)
          return(false);
```

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

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

# Some Hypothetical Method Calls: Condition/Return



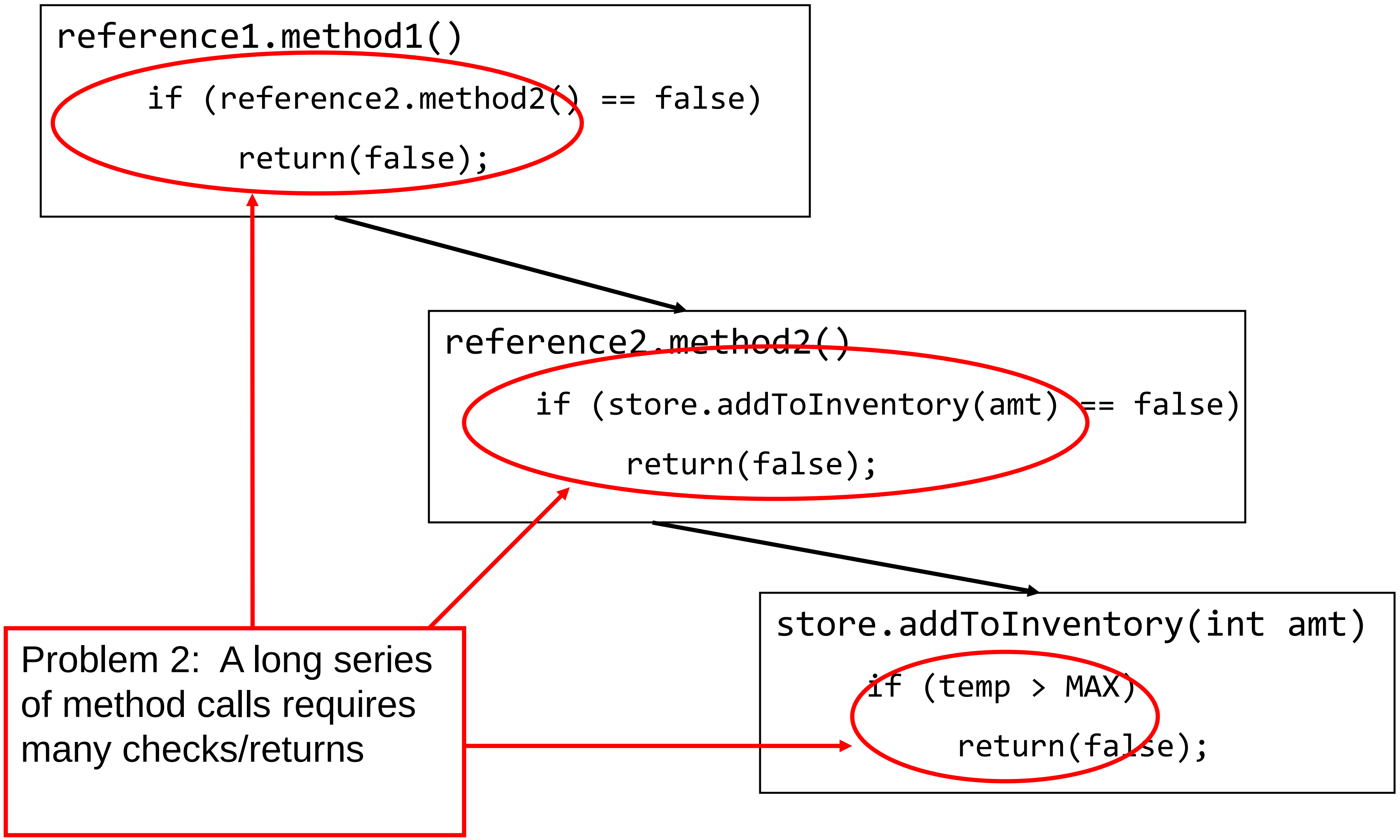
```
reference1.method1()
    if (reference2.method2() == false)
        return(false);
```

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

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

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

# Some Hypothetical Method Calls: Condition/Return



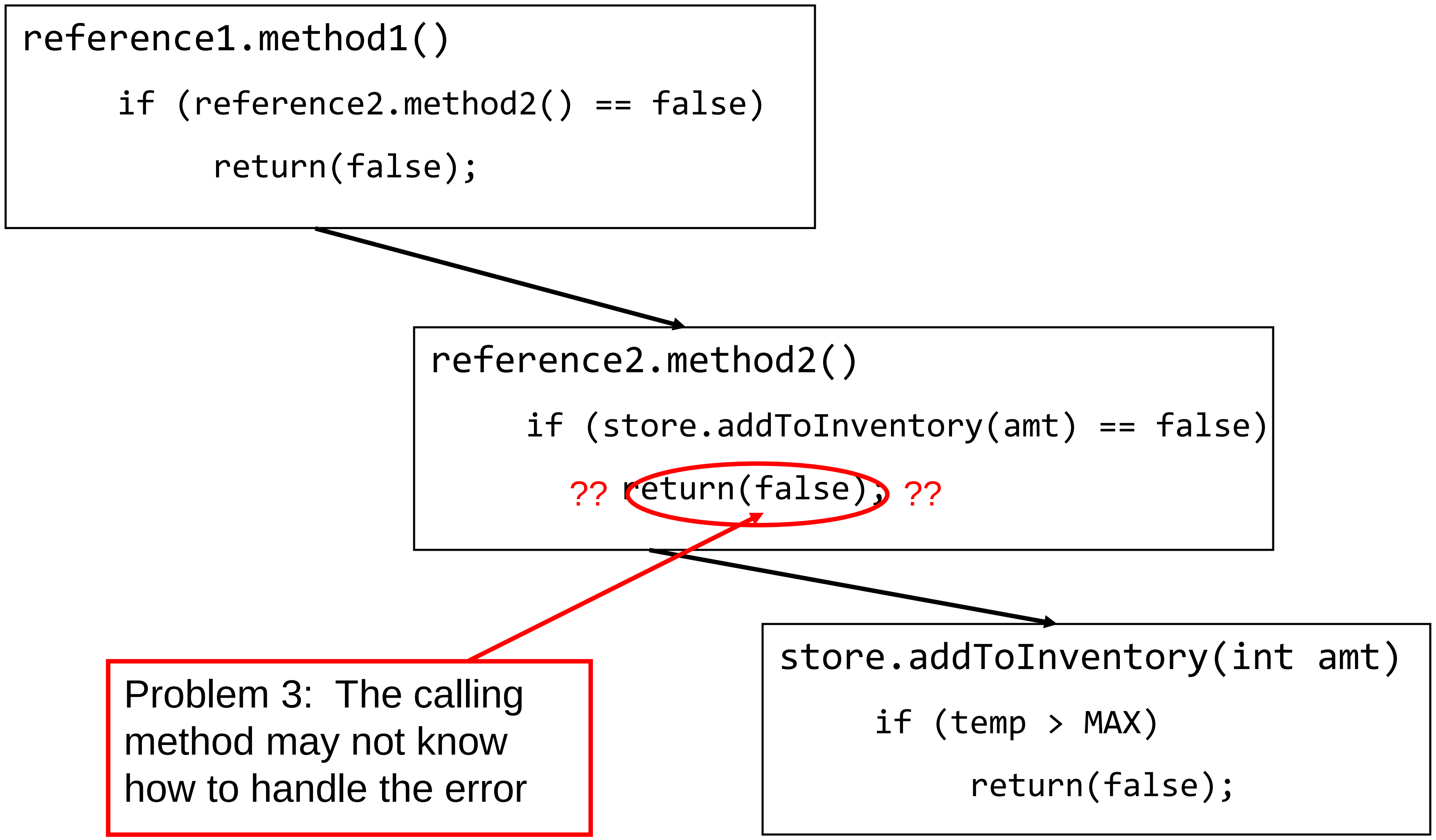
```
reference1.method1()
      if (reference2.method2() == false)
            return(false);
```

```
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

# 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

## 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

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);
 }
```

## 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 ()`

The second 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);
 }
```

## Where The Exceptions Occur In Class Integer

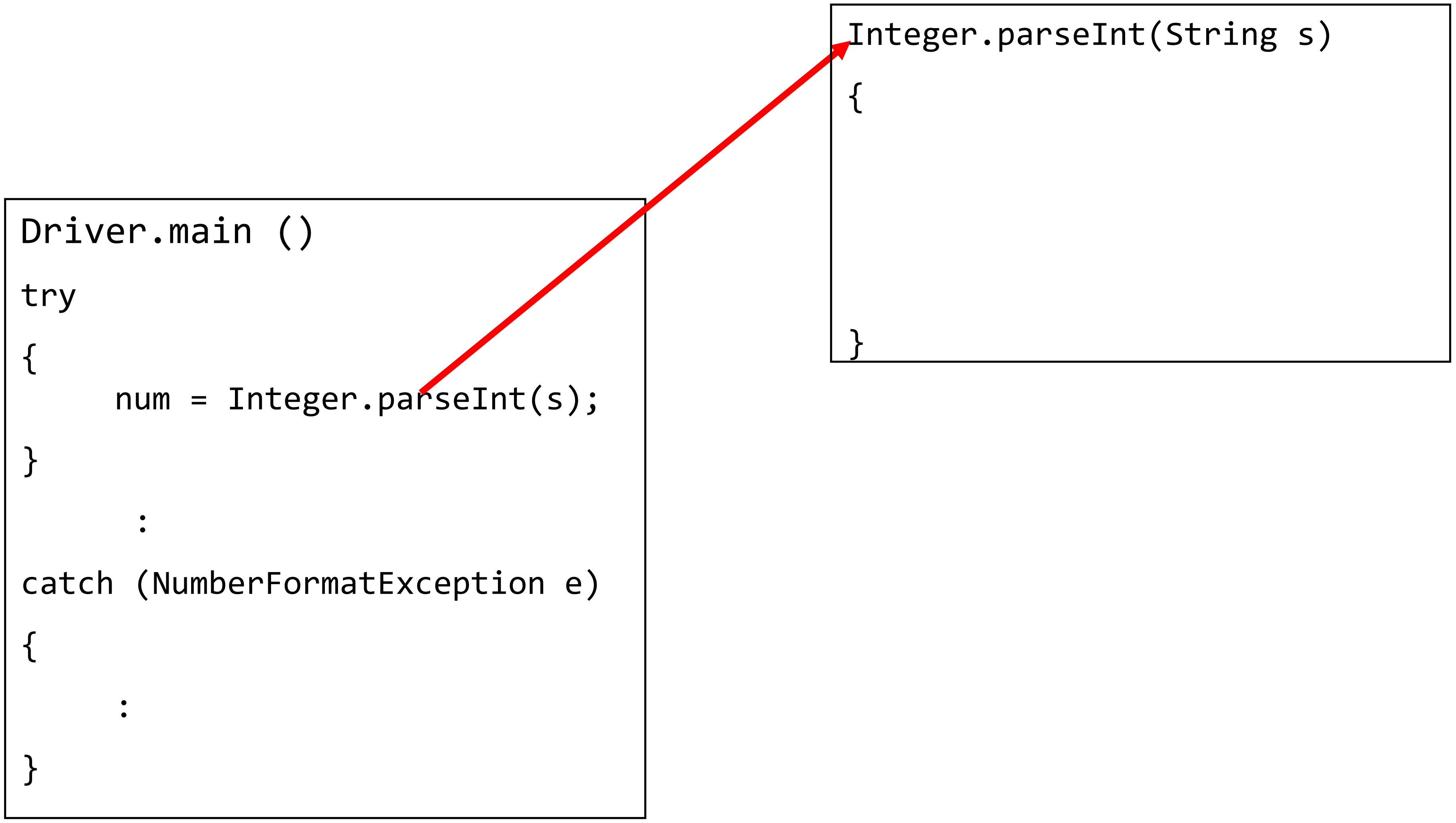
- For online documentation for this class go to:
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

```
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



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

```
Integer.parseInt(String s)
{



}
```

# Handling Exceptions: Tracing The Example

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

}
```

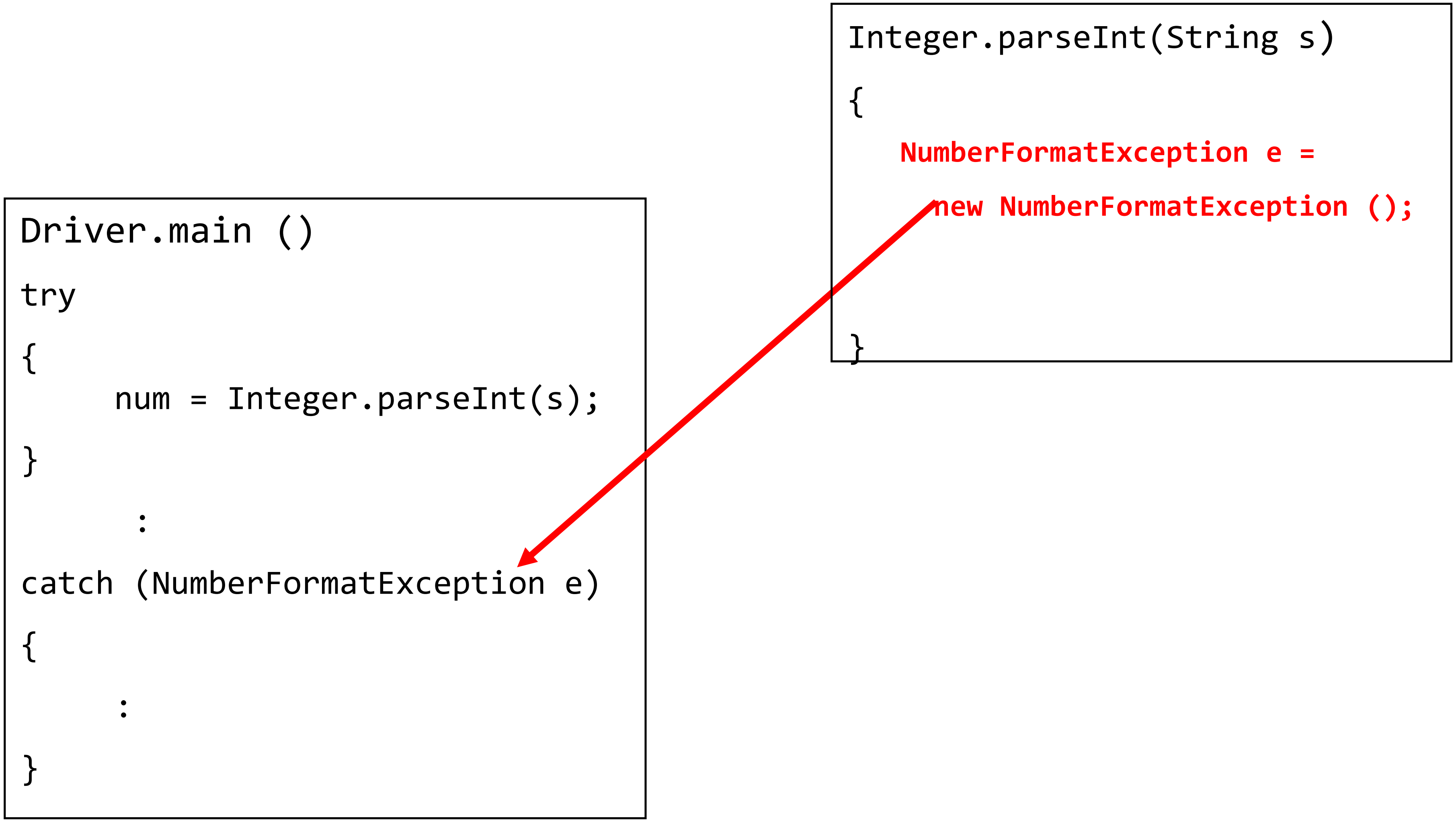
```
Driver.main ()
try
{
     num = Integer.parseInt(s);
}
      :
catch (NumberFormatException e)
{
     :
}
```

## Handling Exceptions: Tracing The Example

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


}
```

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

# Handling Exceptions: Tracing The Example



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


}
```

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

# 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.NumberFormatExceptio
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
}
```

NO!

## 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

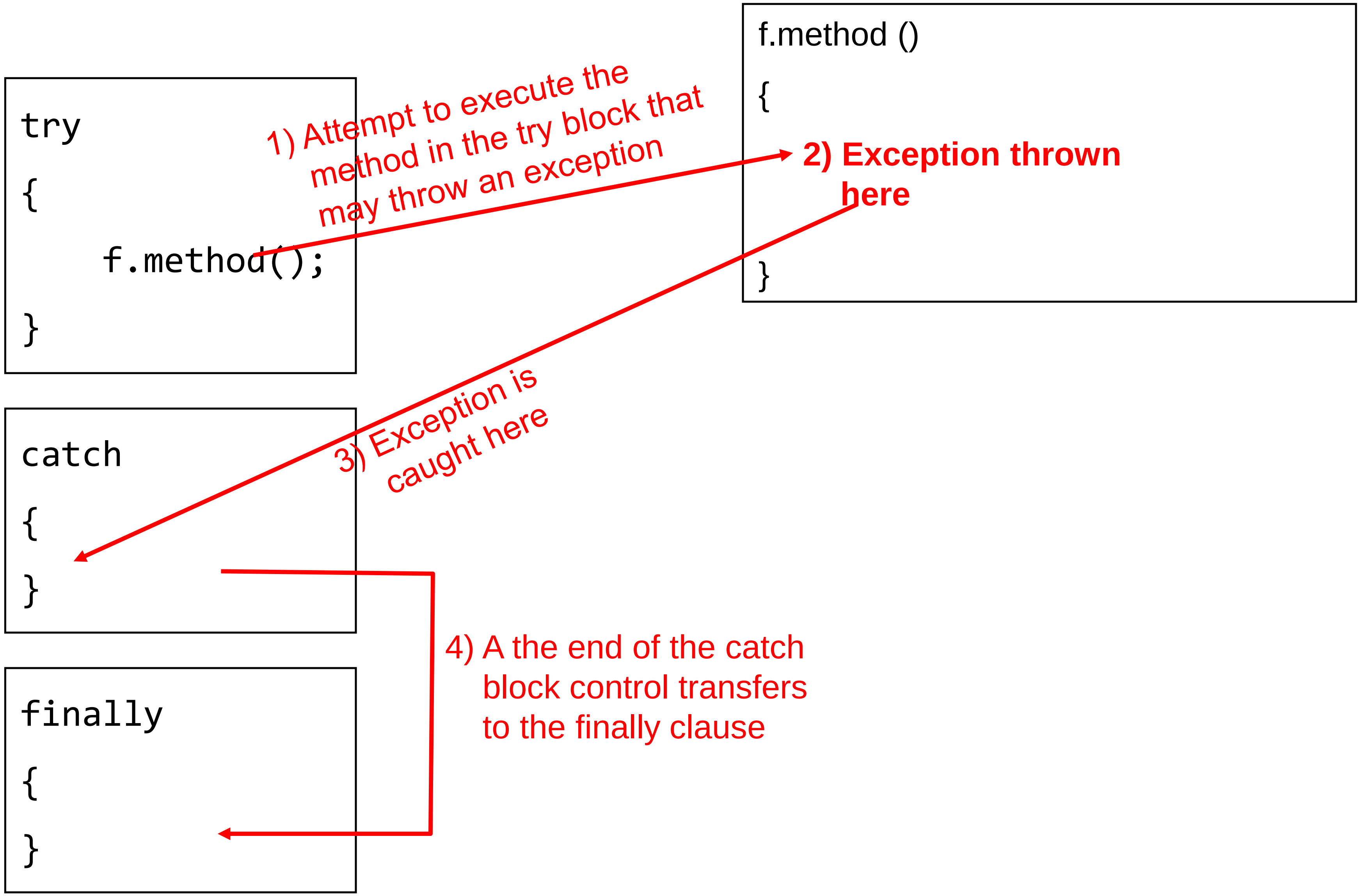
```
try
{
    f.method();
}
```

```
catch
{
}
```

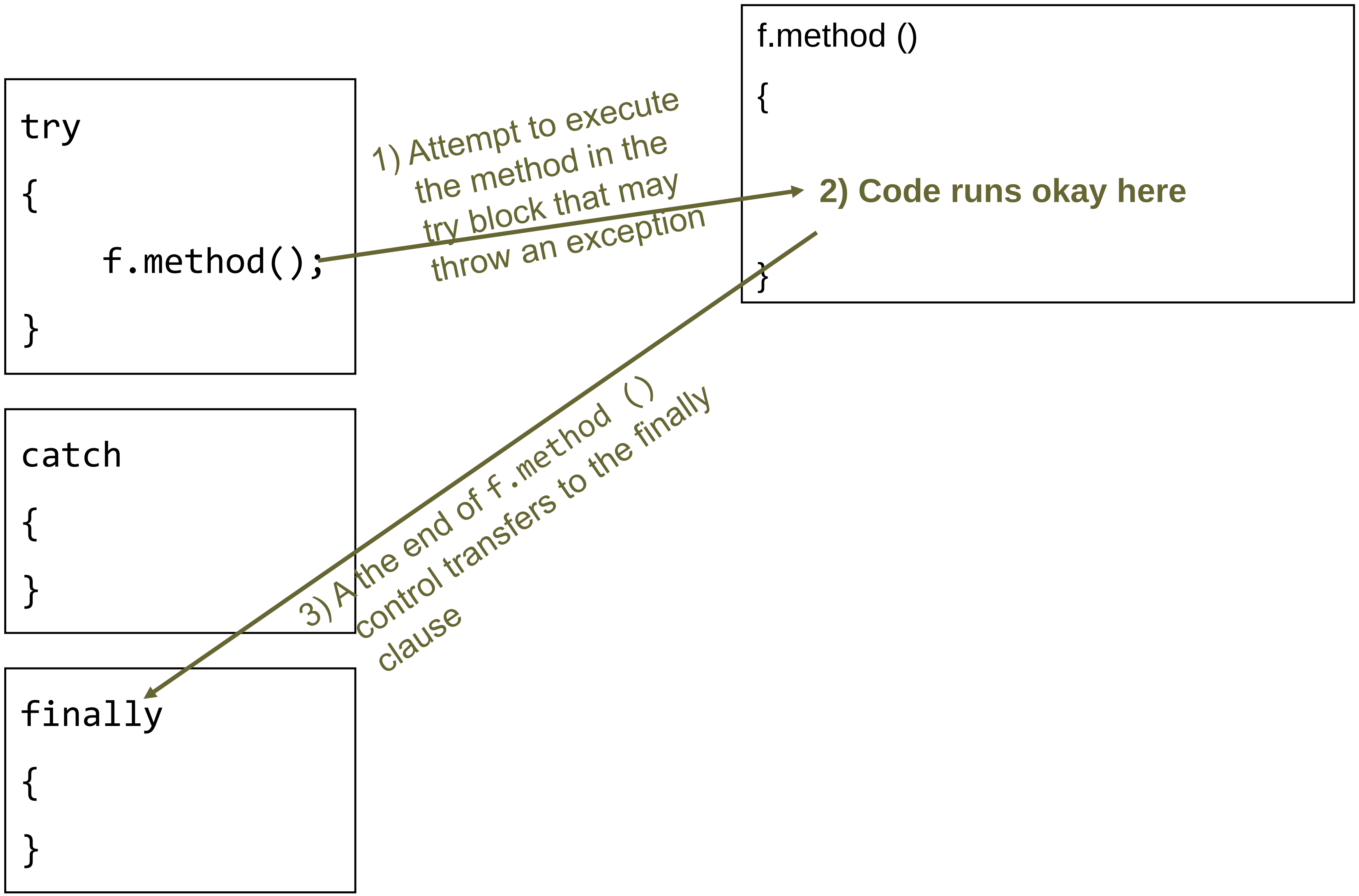
```
finally
{
}
```

```
f.method ()
{


}
```

# The `Finally` Clause: Exception Thrown


try
{
f.method();
}
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

# 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) At the end of f.method () control transfers to the finally clause

## 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();
    }
}
```

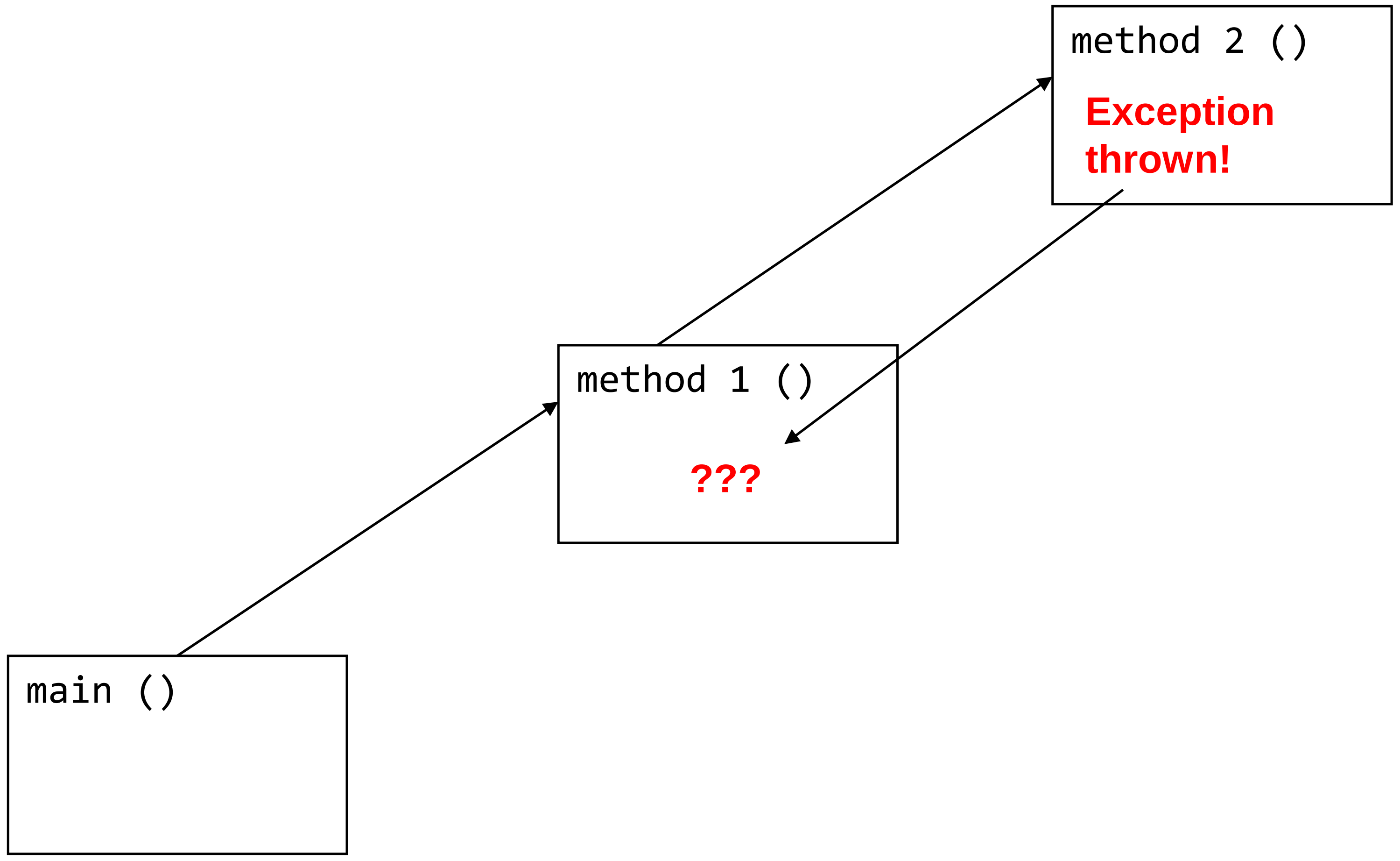
## Try-Catch-Finally: An Example (2)

```
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



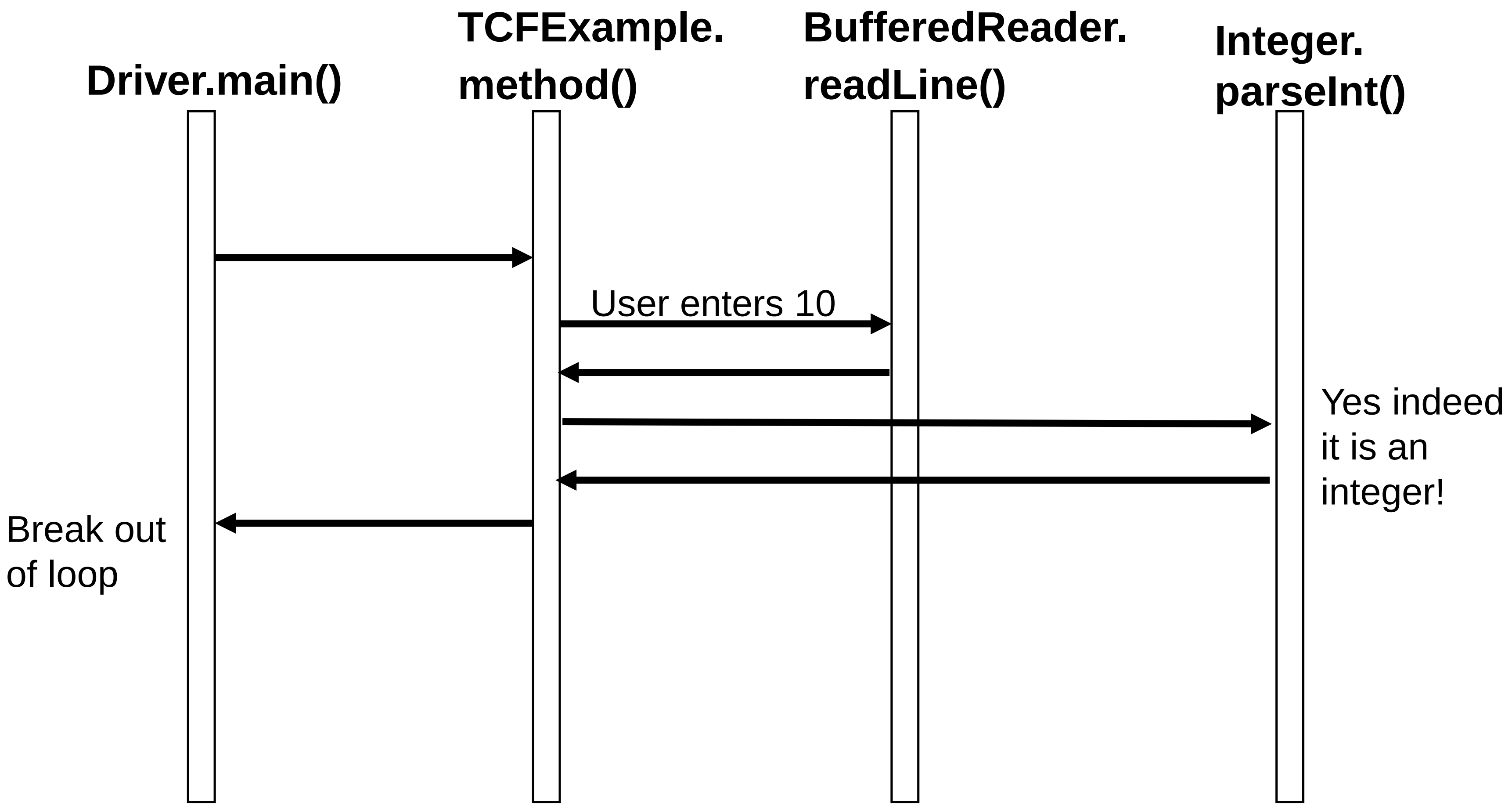
## When The Caller Can’t Handle The Exceptions: An Example

Location of the online example:

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

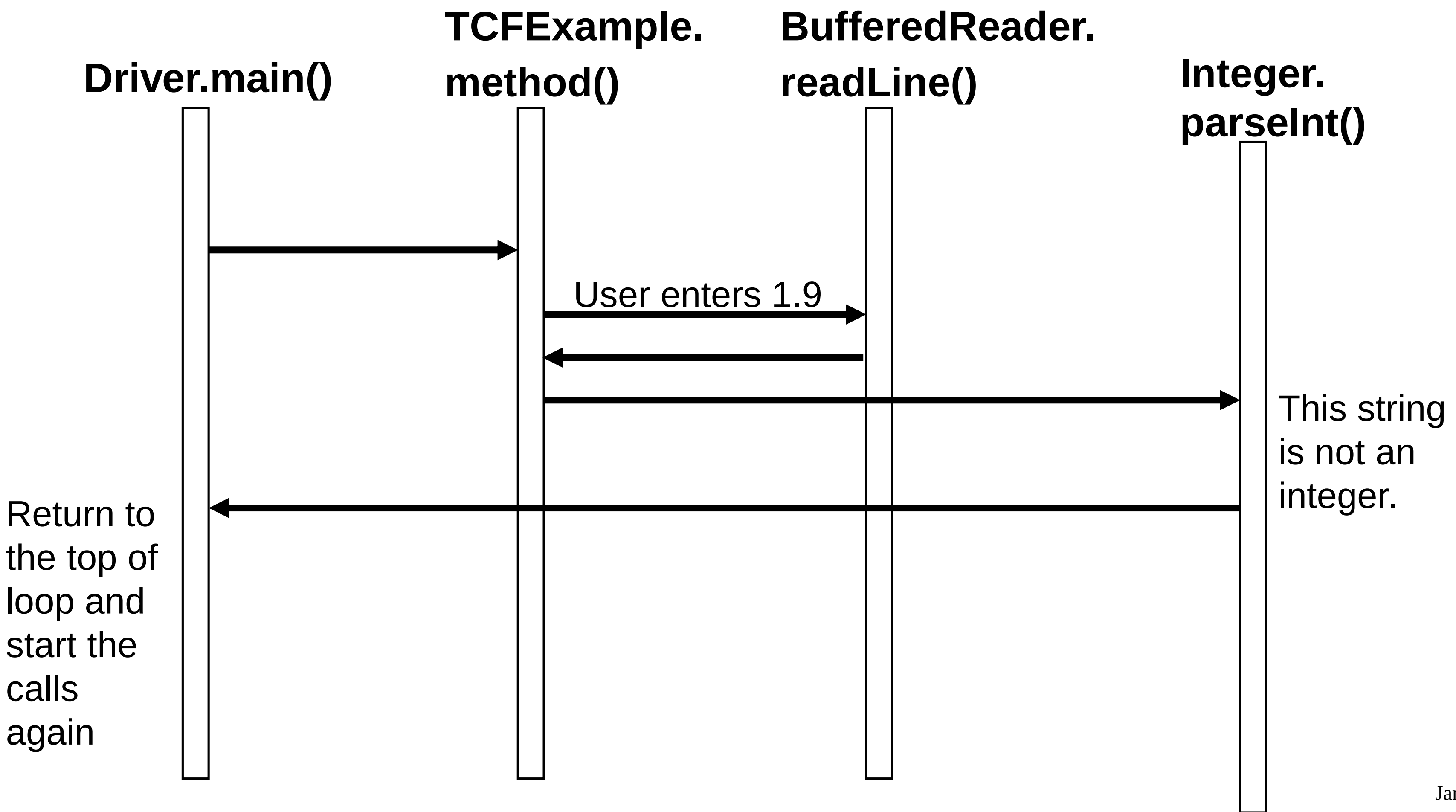
## When The Caller Can’t Handle The Exceptions: An Example (2)

- 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
```

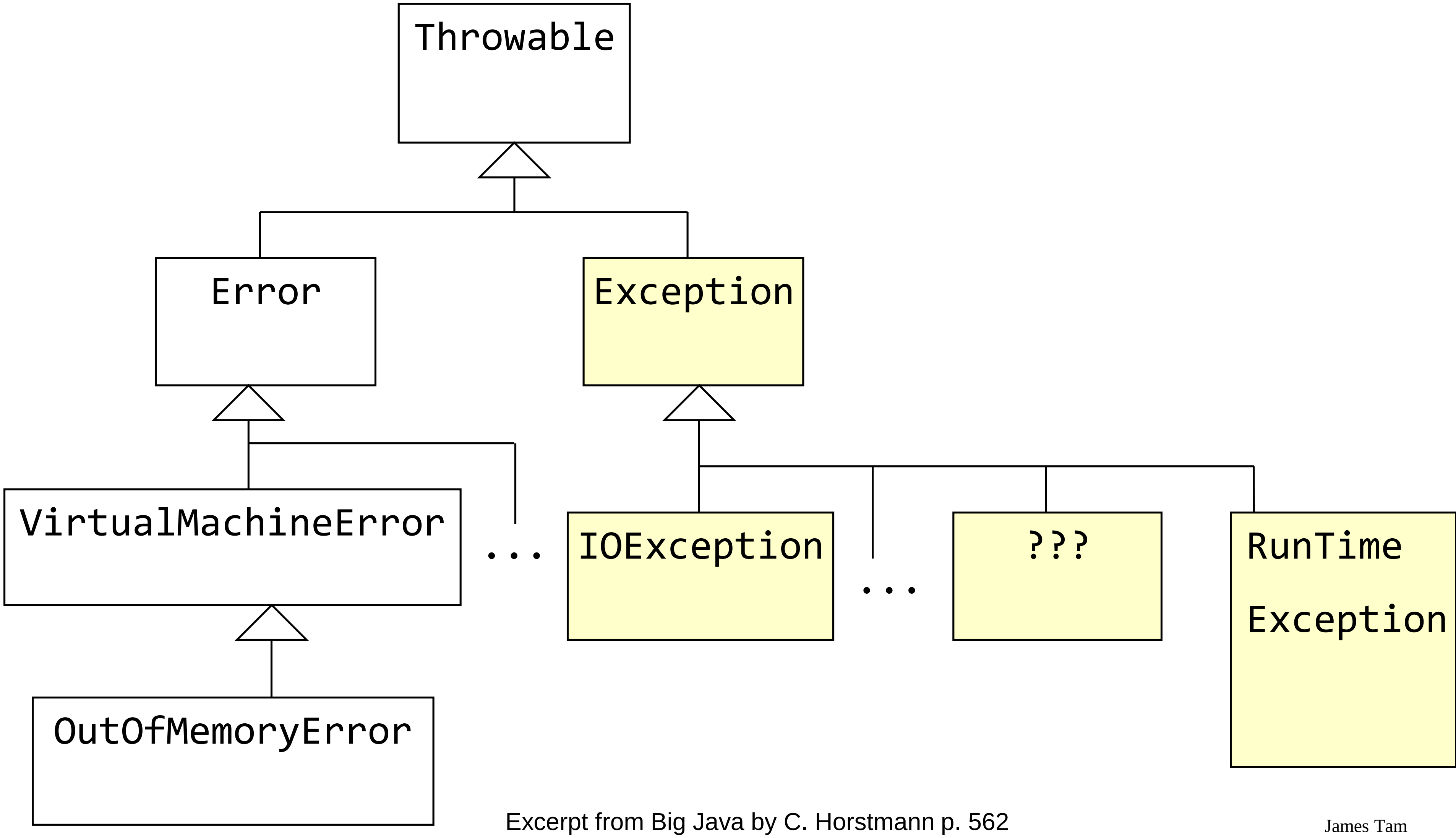
# When The Caller Can’t Handle The Exceptions: An Example (6)

```
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

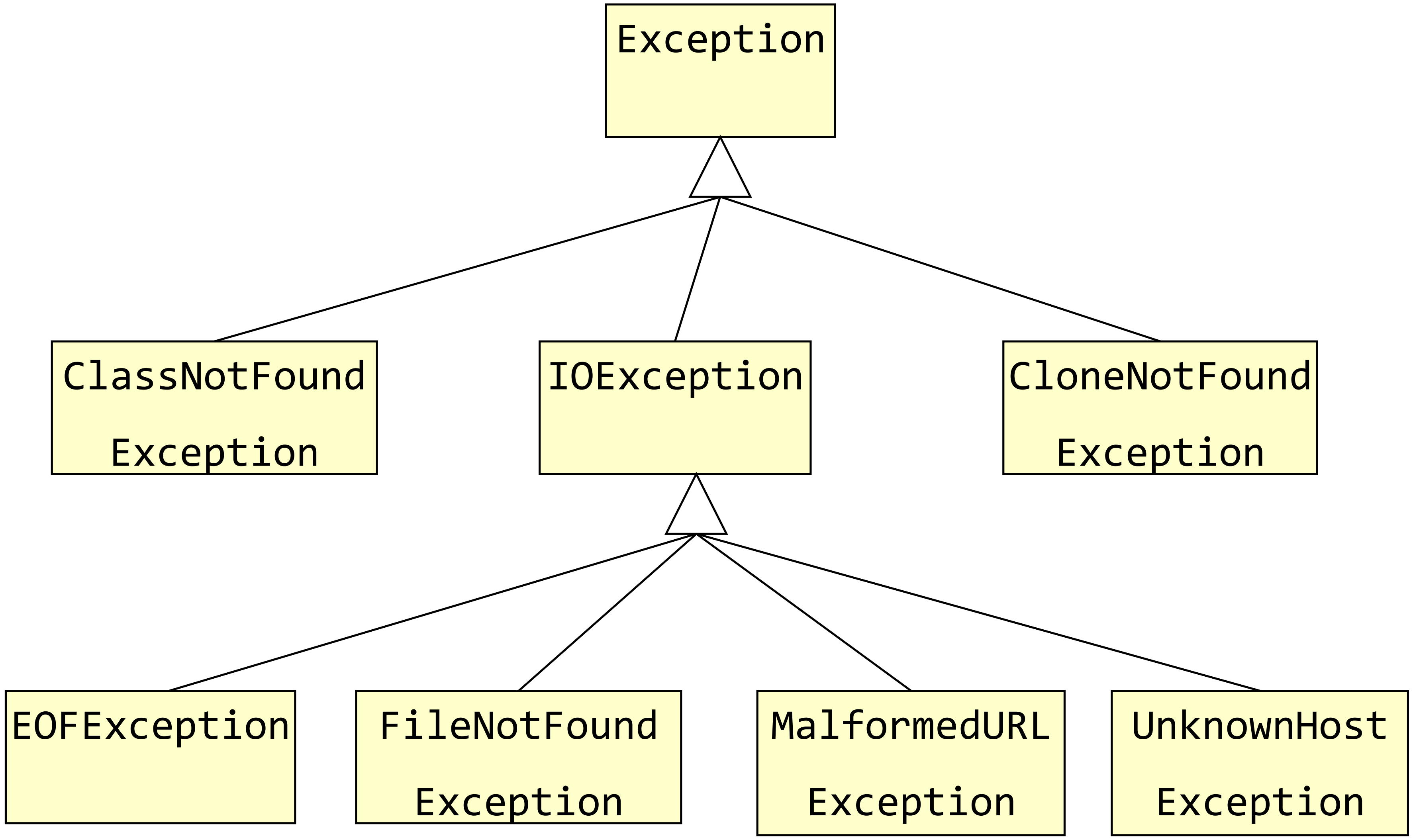
```
public class Driver
{
    public static void main(String [] args) throws
      IOException, NumberFormatException
    {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

## Creating Your Own Exceptions (If There Is Time)



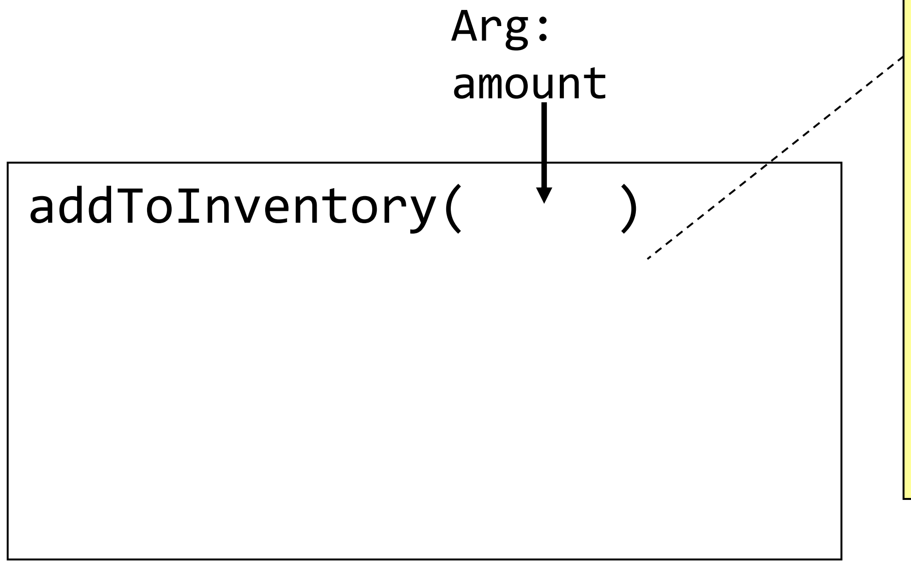
Excerpt from Big Java by C. Horstmann p. 562

# Class Exception: The Local Inheritance Hierarchy



## Writing New Exceptions

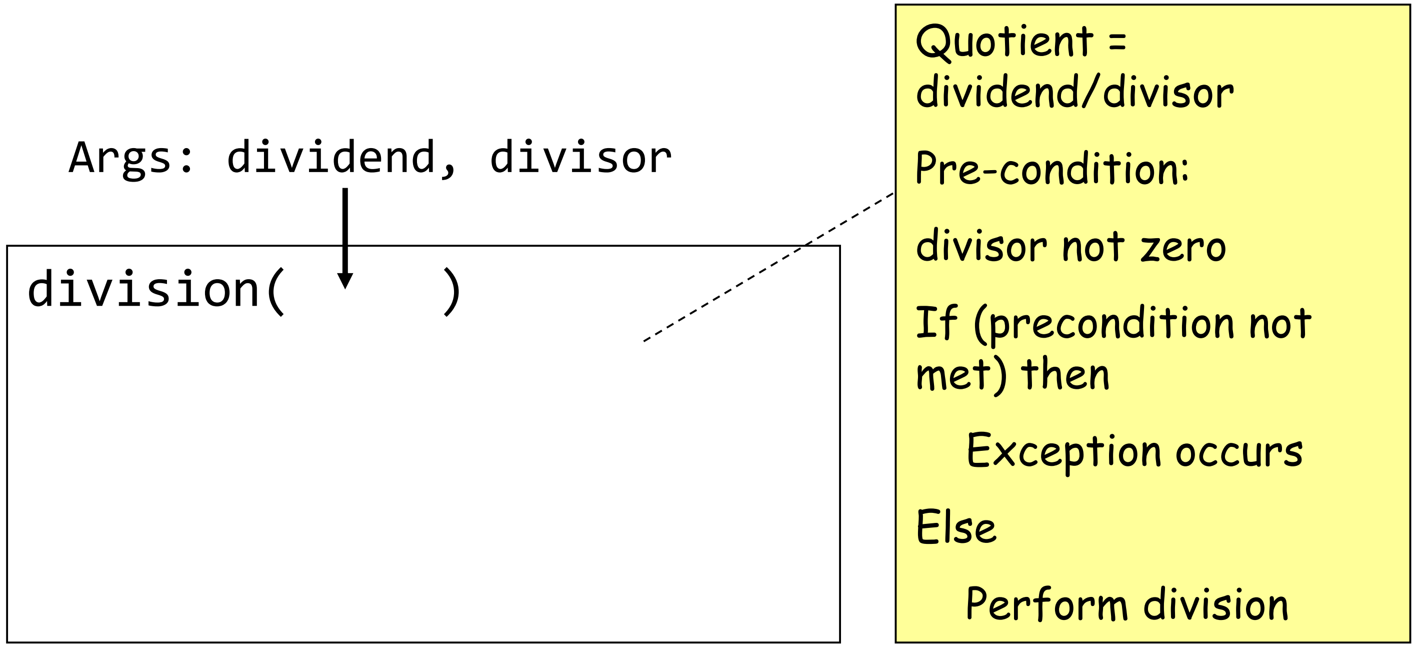
- Typical approach: tie the exception into preconditions
- Remember: preconditions are things that must be true when a function is called.
- Example: Inventory example



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

# 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<<");
        }
```

## 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<<");
}
```

## 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<<");
}
```

## 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());
    }
}
```

# 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);
    }
```

## 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

## 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);
    }
}
```

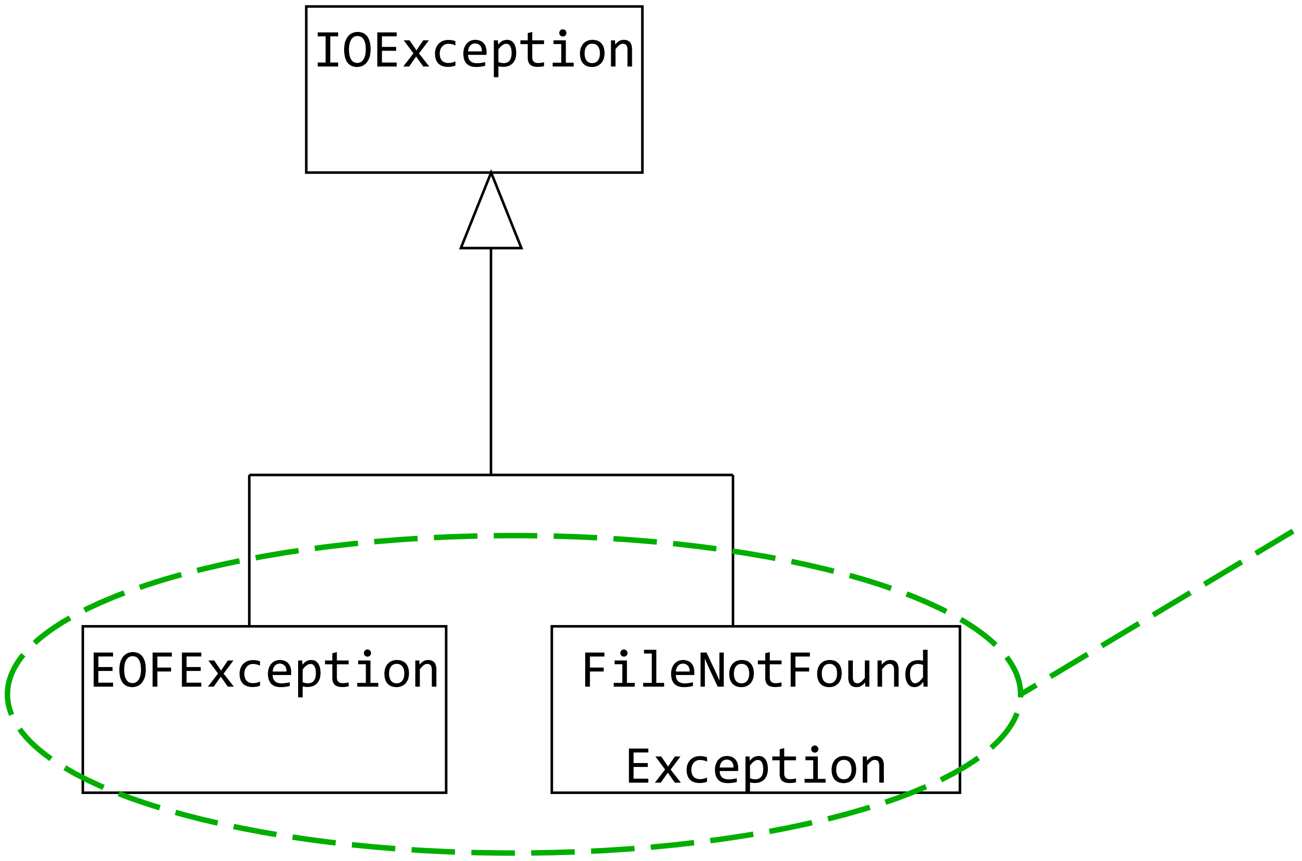
## Writing New Exceptions: Class InventoryOverMaxException

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

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

## Writing New Exceptions: Class InventoryUnderMinException

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

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

## Inheritance Hierarchy For IOExceptions



These classes are more specific instances of class IOException

## 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)
{


}
```

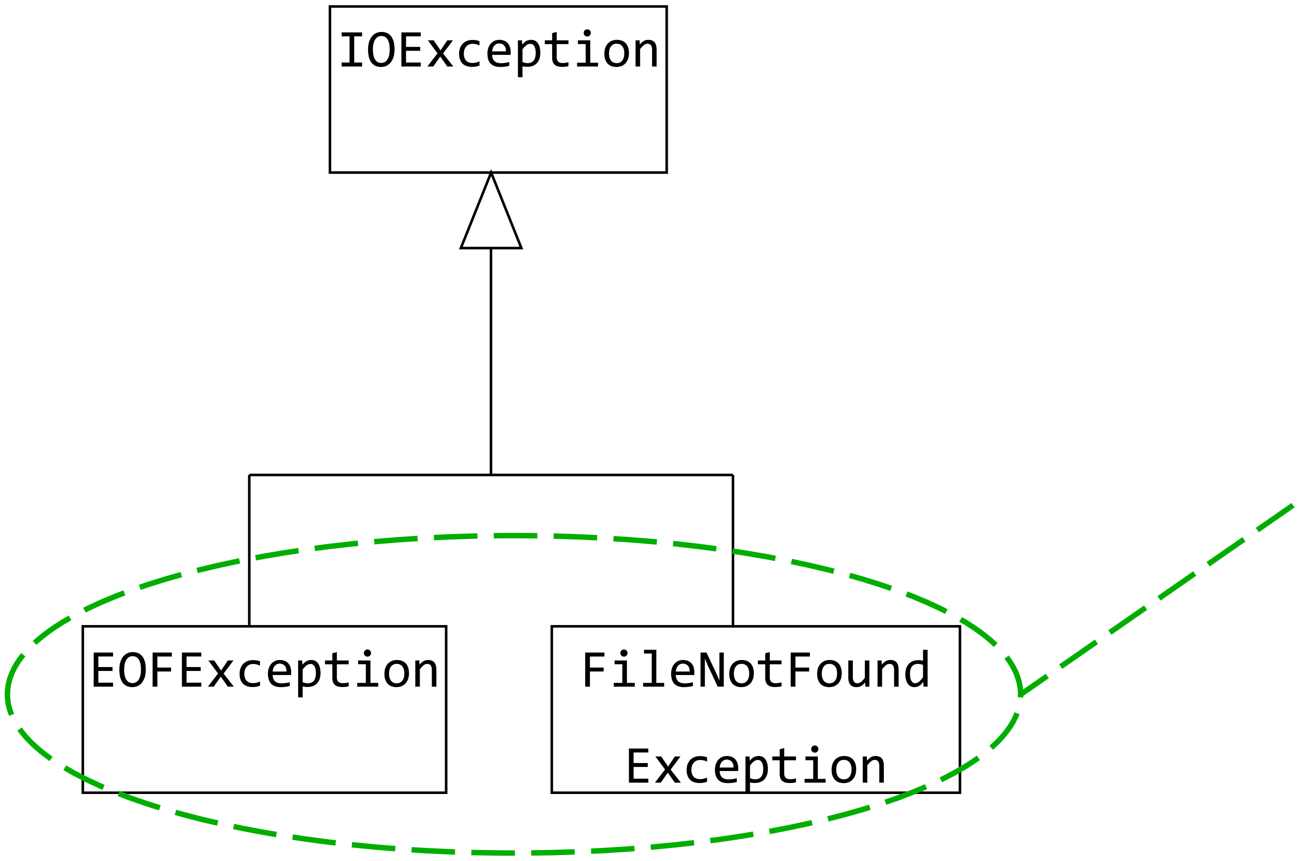
## After This Section You Should Now Know

- 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

## <u>Simple File Input And Output</u>

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

## Inheritance Hierarchy For IOExceptions



These classes are more specific instances of class `IOException`

## 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)

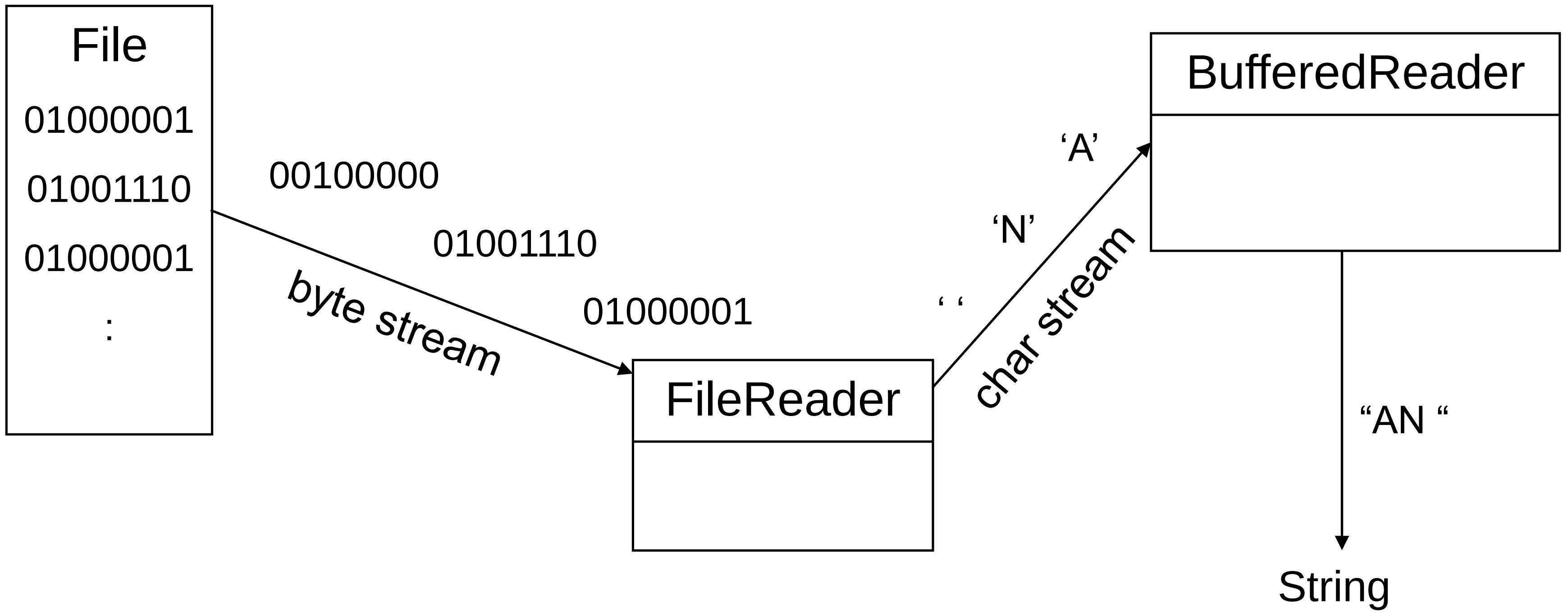
**Correct**

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

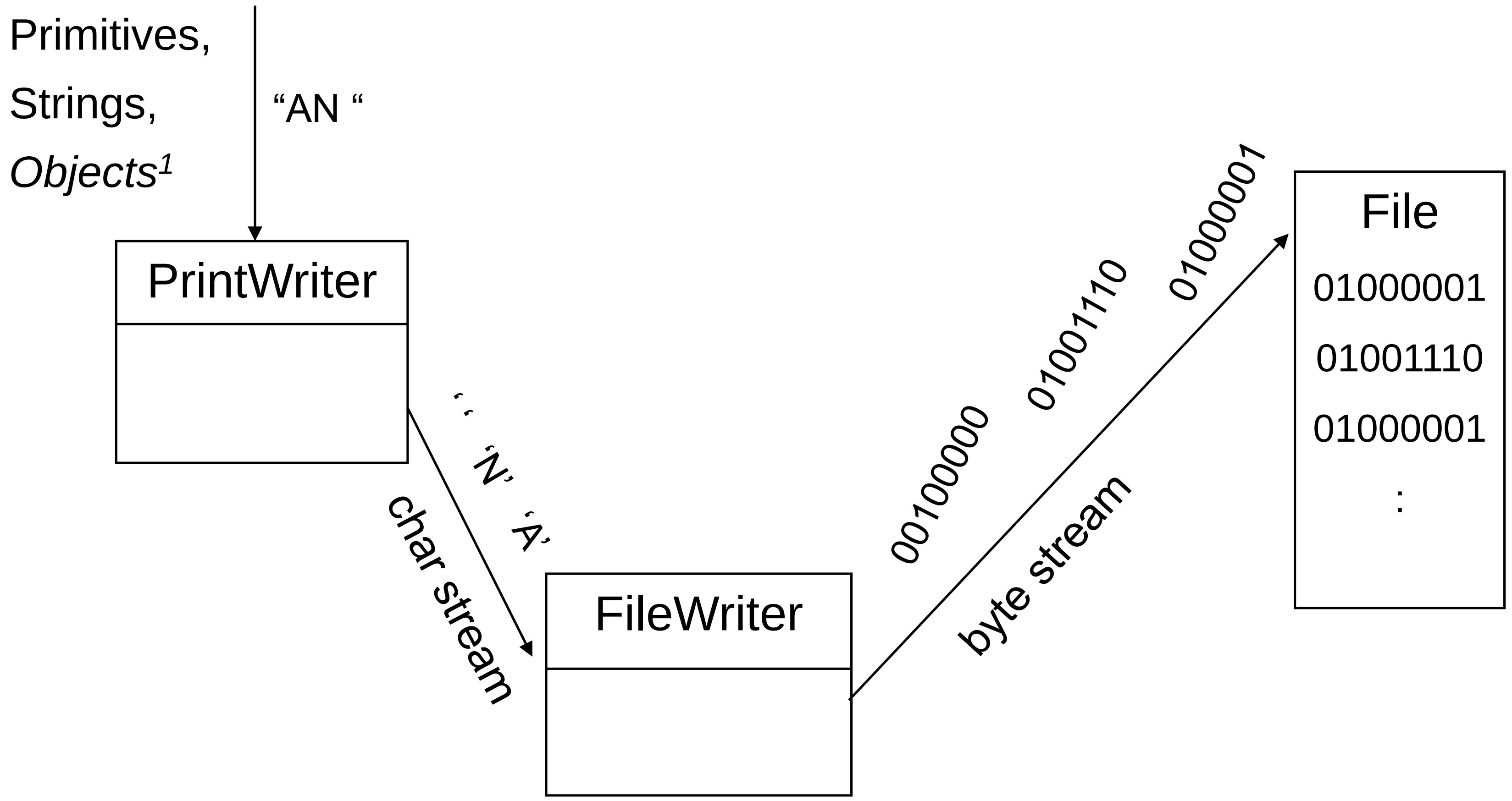
**Incorrect**

```
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];
```

# 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");
}
```

# 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");
 }
```

## After This Section 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`

## Java Packages

- Packages, a method of subdividing a Java program and grouping classes

One source reference:
https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

## Decomposing Object-Oriented Programs Only By Classes

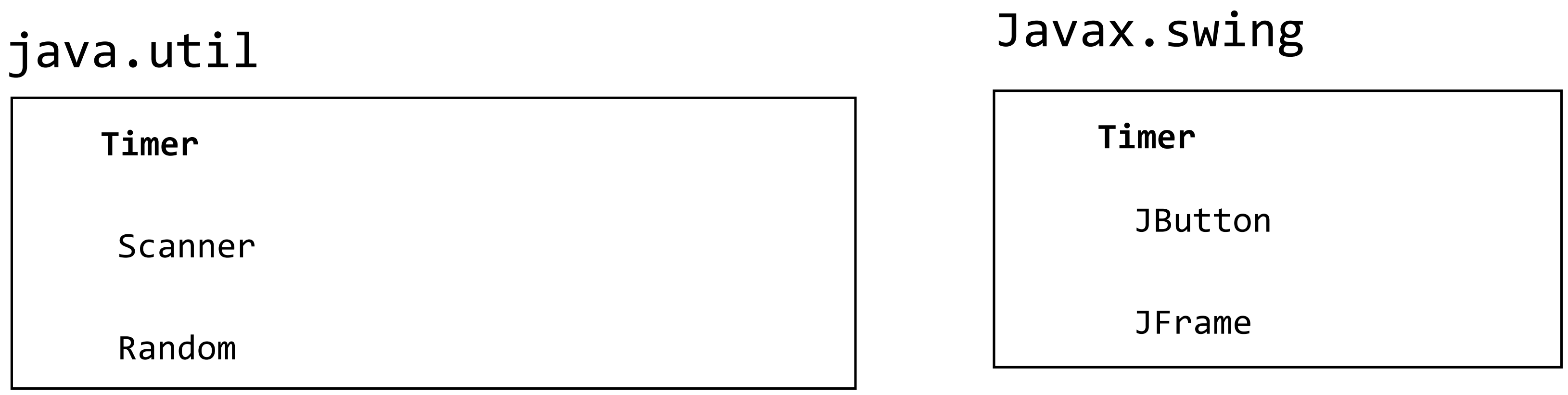
- Works well for small programs e.g., The first problem solving assignment (2D array of references), hierarchies assignment

# Decomposing Larger Object-Oriented Programs

- There is another tool to group related classes, packages.
- **`Java.lang`:** classes that included with the 'core' portion of the Java language:
  - `String`
  - `Math`
  - `:`
- **`Java.util.zip`:** classes that allow for the reading and writing to zip and gzip compressed files:
  - `ZipInputStream`
  - `ZipOutputStream`
  - `:`
- **`Java.awt`:** the original collection of classes used for creating graphical user interfaces:
  - `Button`
  - `Menu`
  - `:`
- **Javax.swing:** the new collection of classes used for creating graphical user interfaces

# Benefits Of Employing Packages

- Increased ease finding a class
- Can be used to prevent naming conflicts



- An additional permission level (package level) may be set to allow certain classes to be instantiated only within the methods of the classes that belong to the same package

## Defining A Package

- Used to group a number of classes together into one related package
- **Format** (done at the top of a class definition)

```
package <package name>;
```

- **Example**:

```
package pack1;
public class IntegerWrapper { ... }
```

## Fully Qualified Names: Includes Package

- $\overbrace{\texttt{pack3}}^{\text{package name}}\texttt{.}\underbrace{\texttt{OpenFoo}}_{\text{class name}}\texttt{.}\underbrace{\texttt{toString}}_{\text{method name}}\texttt{()}$

```
pack3.ClosedFoo.toString()
```

## Importing Packages

- Importing all classes from a package (generally regarded as bad practice because it may allow naming conflicts to occur)

  **Format**

  ```
  import <package name>.*;
  ```

  **Example**

  ```
  import java.util.*;
  ```

- Importing a single class from a package

  **Format**

  ```
  import <package name>.<class name>;
  ```

  **Example**

  ```
  import java.util.Vector;
  ```

## Importing Packages (2)

- When you do not need an import statement:
  - When you are using the classes in the `java.lang` package.
  - You do not need an import statement in order to use classes which are part of the same package

- Excluding the import (from classes other than those from `java.lang`) requires that the full name be provided:

```
java.util.Random generator = new java.util.Random ();
```

Vs.

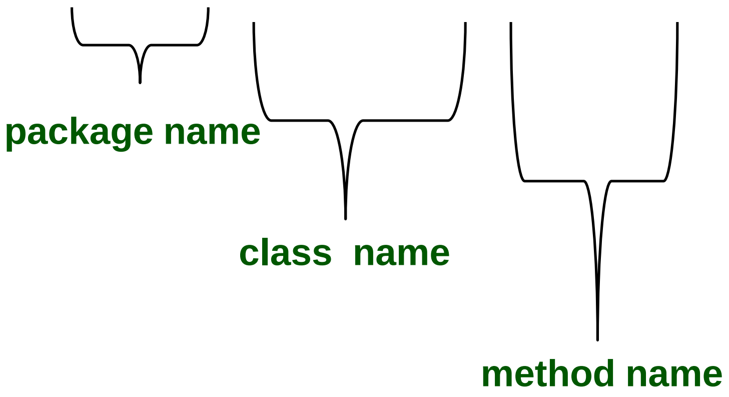
```
import java.util.Random;
Random generator = new Random ();
```

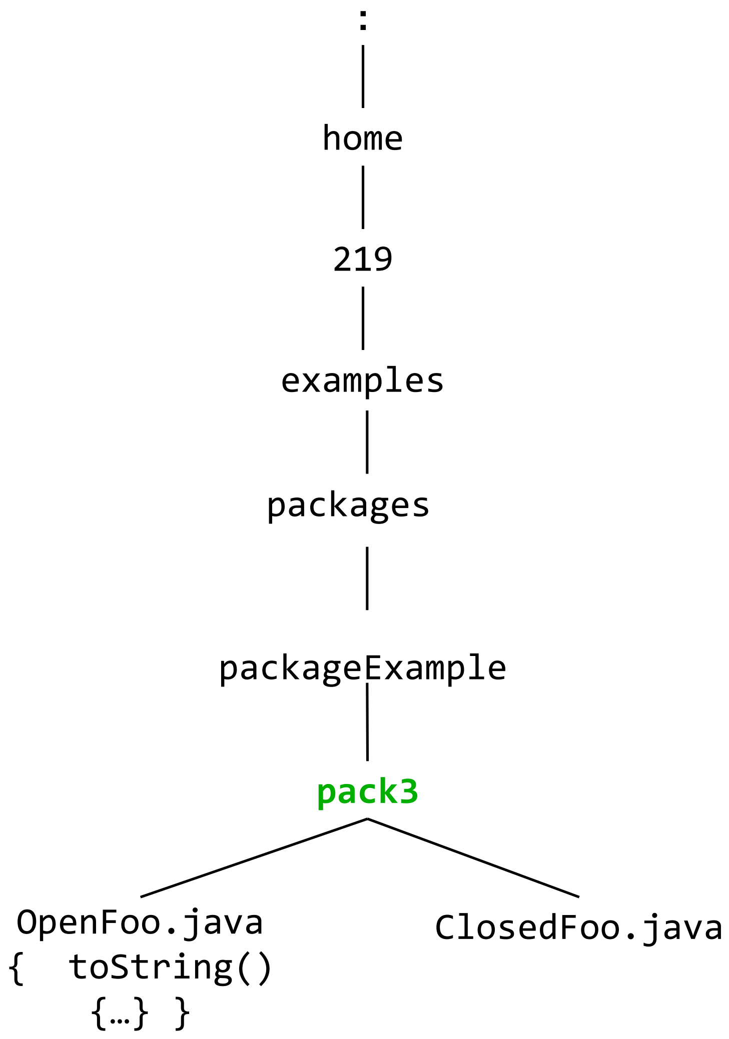
## Default Package

- If you do not use a package statement then the class implicitly becomes part of a default package.
- All classes which reside in the same directory are part of the default package for that program.

# Fully Qualified Names: Matches Directory Structure

- `pack3.OpenFoo.toString()`





# Where To Match Classes To Packages

1. In directory structure: The classes that belong to a package must reside in the directory with the same name as the package (**previous slide**).
2. In the class source code: At the top class definition you must **indicate the package** that the class belongs to.

- **Format:**

```
package <package name>;
<visibility – public or package> class <class name>
{

}
```

## Class Level Access: Public, Package

- **Example** (classes in package ‘`pack3`’)

**`OpenFoo.java`**

```
package pack3;
public class OpenFoo {
       :
}
```

**`ClosedFoo.java`**

```
package pack3;
     class ClosedFoo {
       :
}
```

# Class Level Access: Public, Package (2)

- **Example** (classes in package ‘pack3’)

**OpenFoo.java**

```
package pack3;
public class OpenFoo {
          :
}
```

*Public access: Class can be instantiated by classes that aren’t a part of package pack3*

**ClosedFoo.java**

```
package pack3;
       class ClosedFoo {
          :
}
```

*Package access (default): Class can only be instantiated by classes that are a part of package pack3*

## Sun's Naming Conventions For Packages

- Based on Internet domains (registered web addresses)
- e.g., `www.tamj.com`

```
com.tam .games
    j
          .productivity
```

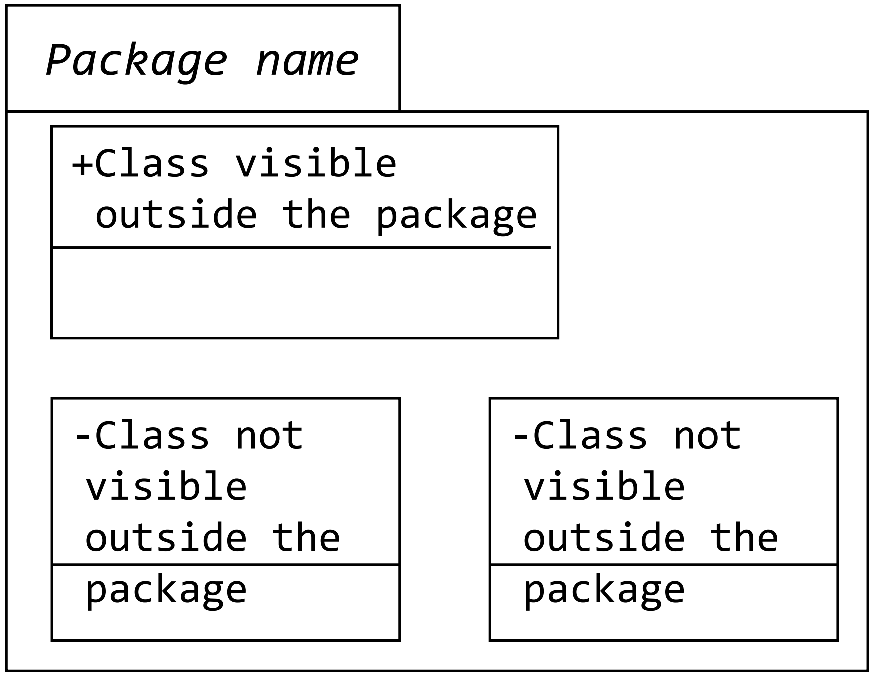
## Sun's Naming Conventions For Packages

- Alternatively it could be based on your email address
- e.g., `tamj@cpsc.ucalgary.ca`

```
ca.ucalgary.cpsc.tamj.games
                     .productivity
```

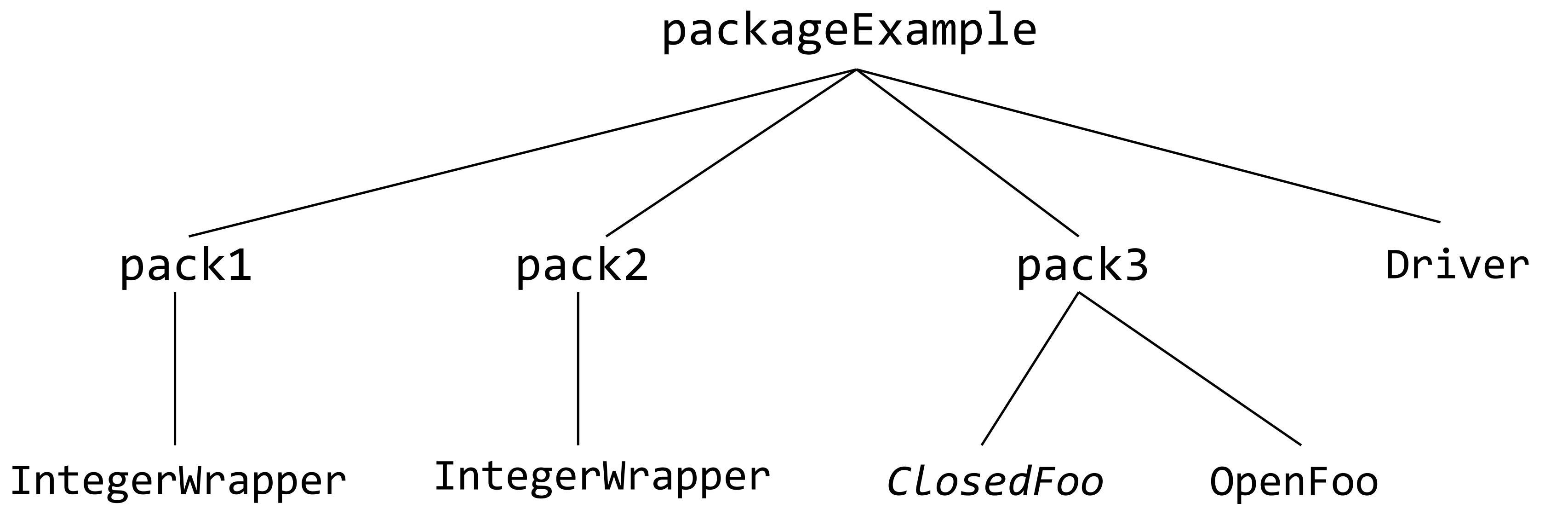
James Tam

# Graphically Representing Packages In UML

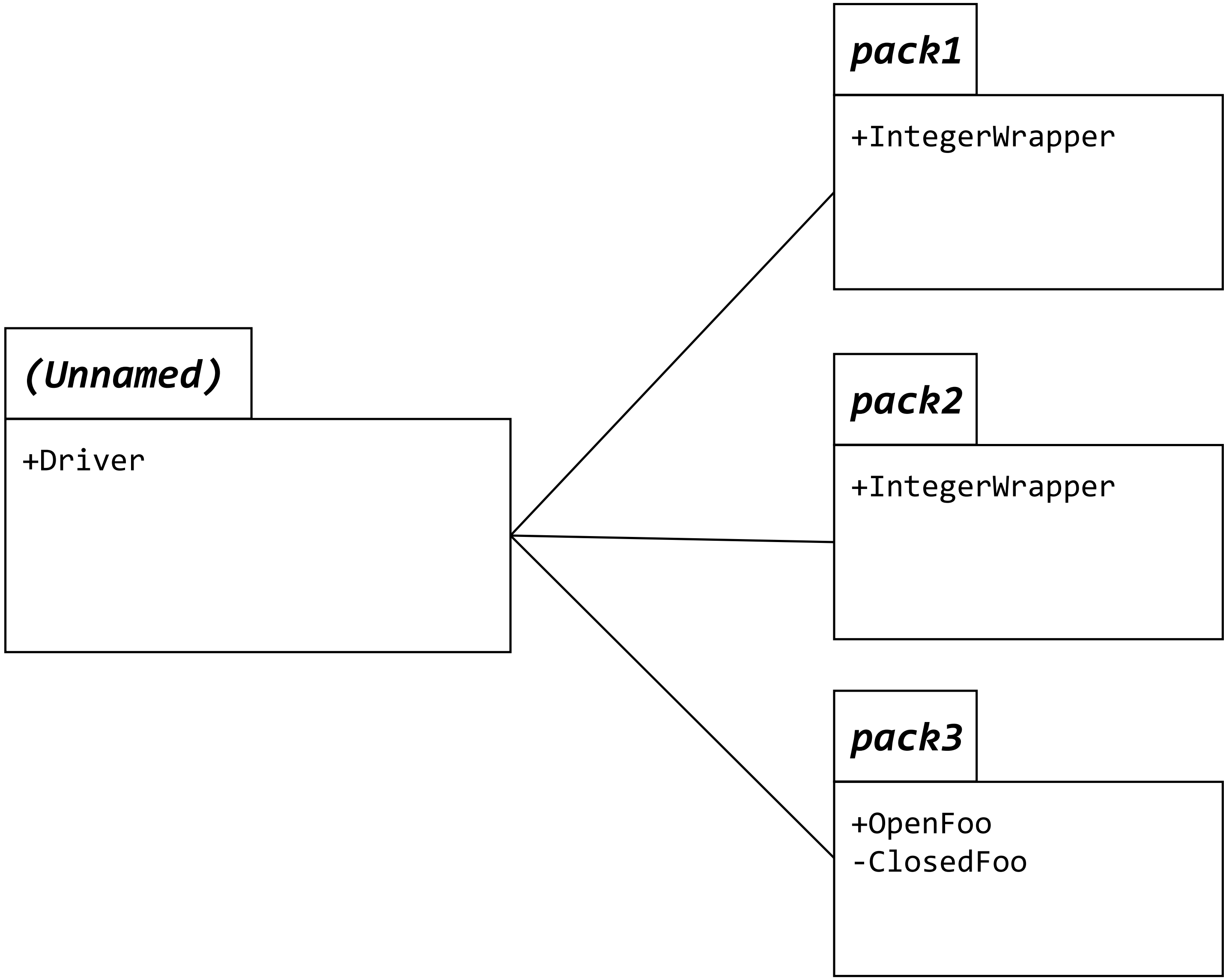

Package name
+Class visible outside the package
-Class not visible outside the package
-Class not visible outside the package

# Packages An Example

- Location of the online example:
- `/home/219/examples/packages/packageExample`
- (But you should have guessed the path from the package name)

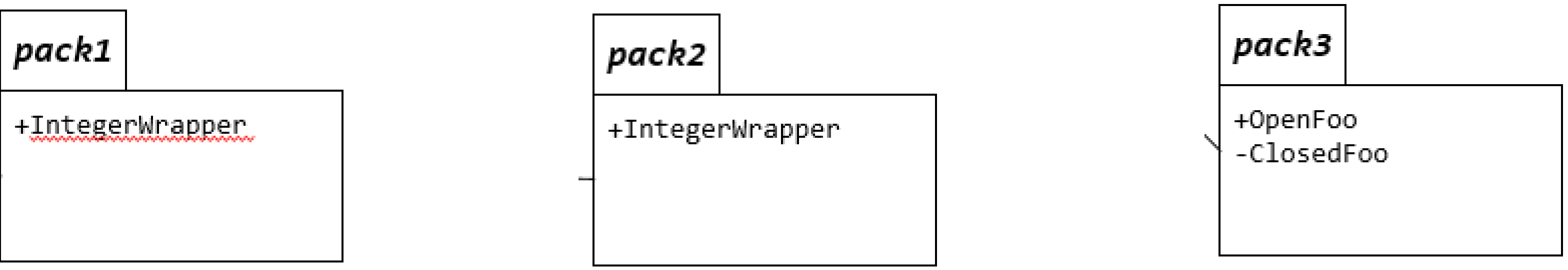


## Graphical Representation Of The Example



# Package Example: The Driver Class

```
import pack3.*;
public class Driver
{
    public static void main(String [] argv)
    {
        pack1.IntegerWrapper iw1 = new pack1.IntegerWrapper();
        pack2.IntegerWrapper iw2 = new pack2.IntegerWrapper();
        System.out.println(iw1);
        System.out.println(iw2);

        OpenFoo of = new OpenFoo ();
        System.out.println(of);
        of.manipulateFoo();
       }
}
```



## Package Example: Package Pack1, Class IntegerWrapper

```
package pack1;
public class IntegerWrapper {

    private int num;

    public IntegerWrapper() {
        num = (int) (Math.random() * 10);
}

    public IntegerWrapper(int newValue) {
        num = newValue;
    }

    public void setNum(int newValue) {
        num = newValue;
    }
```

## Package Example: Package Pack1, Class IntegerWrapper (2)

```
public int getNum() {
    return(num);
}

public String toString() {
    String s = new String ();
    s = s + num;
    return(s);
}
```

## Package Example: Package Pack2, Class IntegerWrapper

```
package pack2;

public class IntegerWrapper {
    private int num;

    public IntegerWrapper() {
        num = (int) (Math.random() * 100);
    }

    public IntegerWrapper(int newValue) {
        num = newValue;
    }

    public void setNum(int newValue) {
        num = newValue;
    }
```

## Package Example: Package Pack2, Class IntegerWrapper (2)

```
    public int getNum() {
        return(num);
    }

    public String toString() {
        String s = new String ();
        s = s + num;
        return(s);
    }
}
```

## Package Example: Package Pack3, Class OpenFoo

```
package pack3;
public class OpenFoo {
    private boolean bool;
    public OpenFoo() { bool = true; }
    public void manipulateFoo() {
        ClosedFoo cf = new ClosedFoo ();
        System.out.println(cf);
    }
    public boolean getBool() { return bool; }
    public void setBool(boolean newValue) { bool = newValue; }
    public String toString(){
        String s = new String ();
        s = s + bool;
        return(s);
    }
}
```

## Package Example: Package Pack3, Class ClosedFoo

```
package pack3;
class ClosedFoo
{
    private boolean bool;
    public ClosedFoo ()
    {
        bool = false;
    }
    public boolean getBool() { return bool; }
    public void setBool(boolean newValue) { bool = newValue; }
    public String toString()
    {
        String s = new String();
        s = s + bool;
        return(s);
    }
}
```

## Updated Levels Of Access Permissions: Attributes And Methods

- **Private "-"**
  - Can only access the attribute/method in the methods of the class where it's originally defined.
- **Protected "#"**
  - Can access the attribute/method in the methods of the class where it's originally defined or the subclasses of that class or in classes of the same package.
- **Package "~" symbol for this permission level**
  - Can access the attribute/method from the methods of the classes within the same package
  - *For Java: If the level of access (attribute or method) is unspecified in a class definition this is the default level of access*
- **Public "+"**
  - Can access attribute/method anywhere in the program

# Updated Levels Of Access Permissions

| Access level | Accessible to | | | |
|---|---|---|---|---|
| | Same class | Class in same package | Subclass in a different package | Not a subclass, different package |
| Public | **Yes** | **Yes** | **Yes** | **Yes** |
| Protected | **Yes** | **Yes** | **Yes** | No |
| Package | **Yes** | **Yes** | No | No |
| Private | **Yes** | No | No | No |

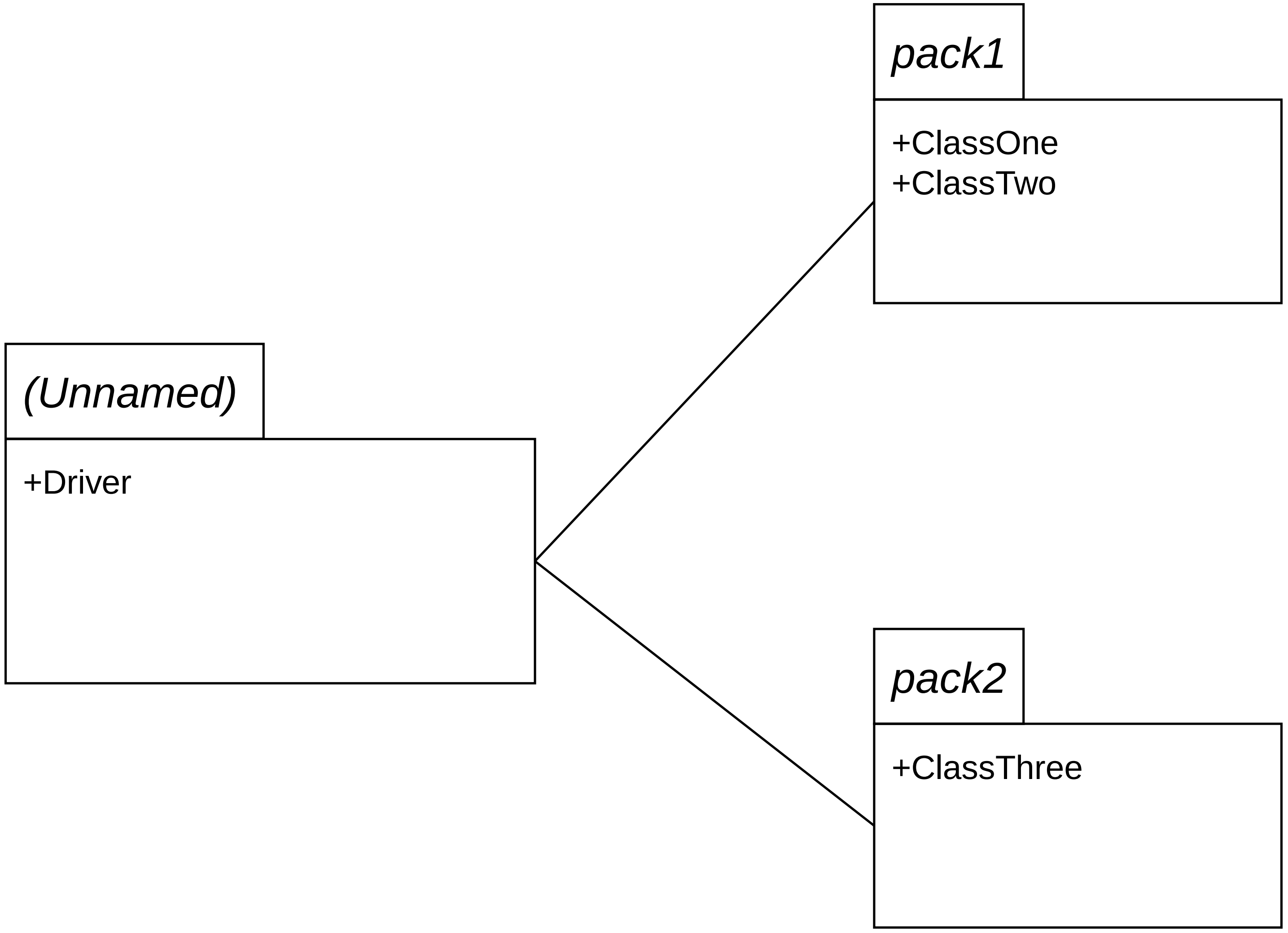
## Updated Levels Of Access Permissions

| | Accessible to | | | |
|---|---|---|---|---|
| Access level | Same class | Class in same package | Subclass in a different package | Not a subclass, different package |
| Public | **Yes: e.g., #1** | **Yes: e.g., #5** | **Yes: e.g., #9** | **Yes: e.g., #13** |
| Protected | **Yes: e.g., #2** | **Yes: e.g., #6** | **Yes: e.g., #10** | No: e.g., #14 |
| Package | **Yes: e.g., #3** | **Yes: e.g., #7** | No: e.g., #11 | No: e.g., #15 |
| Private | **Yes: e.g., #4** | No: e.g., #8 | No: e.g., #12 | No, e.g., #16 |

## Access Permissions: Example

- Location of the example:
- `/home/219/examples/packages/packageExamplePermissions`

# Access Permissions: Examples



# Levels Of Permission, Same Class

- Within the methods of the class, all attributes and methods may be accessed.

```
// Package: pack1
public class ClassOne
{
    public int num1;
    protected int num2;
    int num3;
    private int num4;

    public ClassOne ()
    {
        num1 = 1;   // Example #1
        num2 = 2;   // Example #2
        num3 = 3;   // Example #3
        num4 = 4;   // Example #4
    }
}
```

## Levels Of Permission, Acessible In Class In The Same Package

```
package pack1;
public class ClassOne
{
    public int num1;
    protected int num2;
    int num3;
    private int num4;
}
```

```
package pack1;
public class ClassTwo {
    private ClassOne c1;
    public ClassTwo () {
        c1 = new pack1.ClassOne ();
        c1.num1 = 1;      // Example #5
        c1.num2 = 2;      // Example #6
        c1.num3 = 3;      // Example #7
        // c1.num4 = 4;   // Example #8
    }
}
```

## Levels Of Permission, Subclass In Different Package

```
package pack1;
public class ClassOne
{
    public int num1;
    protected int num2;
    int num3;
    private int num4;
}
```

```
package pack2;
import pack1.ClassOne;
public class ClassThree extends ClassOne
{
    private ClassOne c1;
    public ClassThree ()
    {
        super.num1 = 1;  //Example #9
        super.num2 = 2;  // Example #10
        // super.num3 = 3; // Example #11
        // super.num4 = 4;   // Example #12
    }
}
```

## Levels Of Permission, Not A Subclass, Not In Same Package

```
package pack1;
public class ClassOne
{
    public int num1;
    protected int num2;
    int num3;
    private int num4;
}
```

```
public class Driver {
    public static void main
      (String [] args) {
        pack1.ClassOne c1 = new
          pack1.ClassOne ();
        c1.num1 = 1;         // Example #13
        // c1.num2 = 2;      // Example #14
        // c1.num3 = 3;      // Example #15
        // c1.num4 = 4;      // Example #16
    }
}
```

## After This Section You Should Now Know

- How packages work in Java
  - How to utilize the code in pre-defined packages
  - How to create your own packages
- How the 4 levels of access permission work in conjunction with classes in the same package, sub classes and classes that are neither in the same subclass nor in the same package.