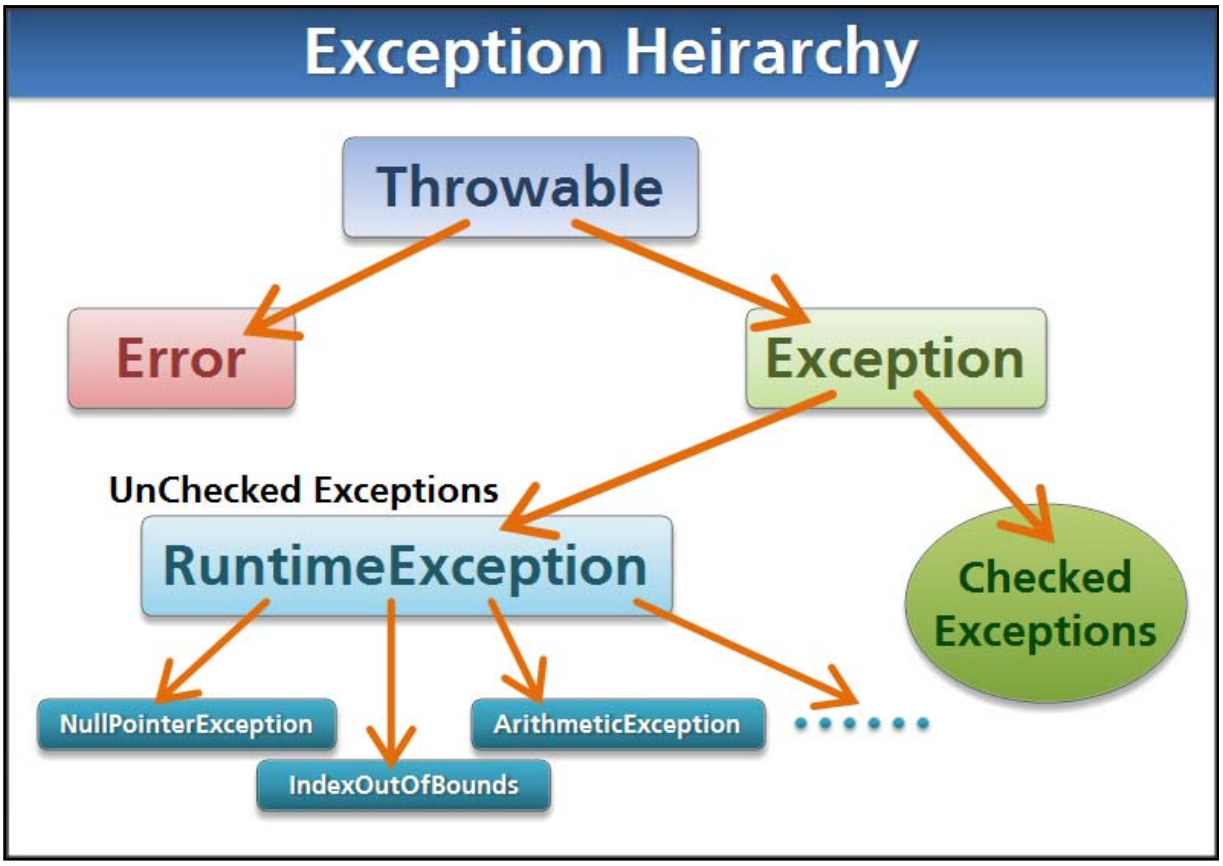
Checked Exceptions and etc.



File I/O using Scanner and PrintStream

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Scanner;


public class FileIO {

    int[] readNumbers(int num)
    {
        int[] arr = new int[num];
        System.out.println("Enter in " + num +" numbers");
        Scanner keyboard = new Scanner(System.in);
        for (int i=0; i < num; i++)
        {
            arr[i] = keyboard.nextInt();
        }
        keyboard.close();
        return arr;
    }


    int[] readNumbers(int num, String filename)
    {
        int[] arr = new int[num];

// In the video, the "input" variable was called keyboard (which is dumb)
        Scanner input = null;
        try {
            FileInputStream fi = new FileInputStream(filename);
            input = new Scanner(fi);

            for (int i=0; i < num; i++)
```

```
            {
                arr[i] = input.nextInt();
            }
        }
        catch (FileNotFoundException e) {
            System.out.println("Error" + e);
        }
        if (input != null)
           input.close();
        return arr;
    }
    void printNumbers(int[] arr)
    {
        for (int i=0; i < arr.length; i++)
            System.out.println(arr[i]);
    }

    void printNumbers(int[] arr, String filename)
    {
        PrintStream ps =null;
        try {
            ps = new PrintStream(filename);

            for (int i=0; i < arr.length; i++)
               ps.println(arr[i]);
        }
        catch (FileNotFoundException e) {
            System.out.println("Error: "+e);
        }

        if (ps != null)
            ps.close();
    }
    public static void main(String[] args) {
        FileIO fio = new FileIO();
        int[] arr = fio.readNumbers(5);
        fio.printNumbers(arr);


        arr = fio.readNumbers(5, "temp.txt");
        fio.printNumbers(arr, "tempOut.txt");

        System.out.println("Program Exitting normally");


    }

}
```

- Take away the try...catch in the above.
- The compile error is because the Exception is checked ... it doesn't have RuntimeException in its derivation path.
- How do you know whether an Exception is checked or unchecked. You can just try it in Eclipse and see if you get a compile error. Or you can look it up in online documentation and take a look at the class derivation heirarchy.
- Demonstrate the "Passing the buck Strategy" in the above example
- Instead of surrounding a section of code with try ... catch, for example, just add:
  `throws FileNotFoundException;`
- Now that we have changed the code so that we can abort this class with an exception, is there any way to guarantee something before we exit? .... **`finally clause is the answer.`**
- For example you successfully open one file, but unsuccessfuly open a second file. When you exit you want to make sure that you have some code that for sure executes to close any files that might still be open:

```
  finally
  {
     System.out.println ("Place to do things that need to be done before exitting .... always");
```

```
    }
```

Ants Game of life code

Game of Life from Arrays_part2

Add the following exception definition

```
class MyException extends RuntimeException
{
 double density;
        MyException(double density)
        {
            super ("Bad Density: "+ density);
            this.density = density;
        }
}
```

Change the Life constructor to:

```
 public Life(double density)
    {
        if (density <= 0.0  || density >= 1.0)
            throw new MyException(density);
        for (int i= 0; i< NUM_ROWS; i++)
            for (int j=0; j < NUM_COLS; j++)
            {
                world[i][j] = new Ant();
                if (Math.random() < density)
                    world[i][j].setAlive(true);
            }
    }
```

Change main to:

```
    public static void main(String[] args)
    {
        System.out.println("Enter in Ant density (number between 0.0 and 1.0)");
        double density = keyboard.nextDouble();
        Life life;
        try {
            life = new Life(density);
            life.playGame(true);
        } catch (MyException e) {
            System.out.println("Caught a MyException: density = "+ e.density);
        }

    }
```

The following notes are not on the video. This documents a subtle detail that I don't plan to cover in class.

If you are interested come see me and we can talk about it.

Overriding methods that throw Exceptions - The derived class can't throw exceptions that are not included in the list of classes thrown by the base class.

```
public class MyBaseClass {
   public void myMethod() throws java.io.IOException { }
}

class OkSubClass0 extends MyBaseClass {
// OK: Derived class throws the same as the base class
```

```
    public void myMethod() throws java.io.IOException {}
 }

 class OkSubClass1 extends MyBaseClass {
   // OK: Derived class throws Exception derived from the
   // Exception thrown by base class
   public void myMethod() throws java.io.FileNotFoundException {}
}

class OkSubClass2 extends MyBaseClass {
   //OK: Derived class doesn't need to throw exceptions
   // even if base class throws exceptions
   public void myMethod() {}
}

class OkSubClass3 extends MyBaseClass {
// OK: All of these Exceptions are derived from IOException
// which is the Exception thrown by the base class
   public void myMethod() throws java.io.EOFException,
      java.io.FileNotFoundException, java.util.zip.ZipException {}
}

class NotOkSubClass extends MyBaseClass {
   // NOT OK: AWTException not derived from IOException
   public void myMethod() throws java.awt.AWTException {} //Illegal
}
```

-

Last Updated: July 28, 2014 10:28 AM