COMP 401
EXCEPTIONS-REMAINDER

Instructor: Prasun Dewan
public static void main(String args[]) {
    try {
        echoLines(numberOfInputLines(args));
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Did not enter an argument. Assuming a single input line.");
        try {
            echoLines(1);
            throw new AMissingArgumentException("Missing first argument");
        } catch (IOException ioe) {
            System.out.println("Did not input the one input string, which is the default in case of missing argument, before input was closed. ");
        } catch (IOException e) {
            System.out.println("Did not input the correct number of input strings before input was closed. ");
        }
    }
}
```java
public static void main(String args[]) {
    long startTime = System.currentTimeMillis();
    try {
        echoLines(numberOfInputLines(args));
        System.out.println(System.currentTimeMillis() - startTime);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Did not enter an argument. Assuming a single input line.");
        try {
            echoLines(1);
            System.out.println(System.currentTimeMillis() - startTime);
            throw new AMissingArgumentException("Missing first argument");
        } catch (IOException ioe) {
            System.out.println("Did not input the one input string, which is the default in case of missing argument, before input was closed.");
            System.out.println(System.currentTimeMillis() - startTime);
        }
    } catch (IOException e) {
        System.out.println("Did not input the correct number of input strings before input was closed.");
        System.out.println(System.currentTimeMillis() - startTime);
    }
}
```
public static void main(String args[]) {
    long startTime = System.currentTimeMillis();
    try {
        echoLines(numberOfInputLines(args));
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Did not enter an argument. Assuming a single input line.");
        try {
            echoLines(1);
            throw new AMissingArgumentException("Missing first argument");
        } catch (IOException ioe) {
            System.out.println("Did not input the one input string, which is the default in case of missing argument, before input was closed.");
            System.out.println(System.currentTimeMillis() - startTime);
        }
    } catch (IOException e) {
        System.out.println("Did not input the correct number of input strings before input was closed.");
    }
    finally {
        System.out.println(System.currentTimeMillis() - startTime);
    }
}
public static int factorial(int n) {
    System.out.println("Started factorial: "+ n);
    try {
        if (n <= 1) return 1;
        return n * factorial(n - 1);
    }
    finally {
        System.out.println("Ended factorial: "+ n);
    }
}
Finally for PostConditions and Invariants

```java
public boolean preSetWeight (double newWeight) {
    return newWeight > 0;
}
public void setWeight(double newWeight) {
    assert preSetWeight(newWeight);
    try {
        if (!preSetWeight(newWeight)) return;
        weight = newWeight;
    } finally {
        assert preSetWeight(newWeight); // invariant - post and pre condition
    }
}
```
INTRA-METHOD PROPAGATION

```java
static Object[] list = {5, "hello", "goodbye"};

for (int i = 0; i < list.length; i++) {
    try {
        System.out.println((String) list[i]);
    } catch (ClassCastException e) {
        System.out.println(e);
    }
}
```

Println terminated, catch executed, and loop continues

```
try{
    for (int i = 0; i < list.length; i++) {
        System.out.println((String) list[i]);
    }
} catch (ClassCastException e) {
    System.out.println(e);
}
```

println terminated and exception propagated to enclosing loop, which is also terminated, and catch executed
**Terminating Program vs. Continuing**

- Independent errors can be collected
  - Scanning: `int 5a = 50`
- Dependent errors cannot be:
  - Parsing: `5 + 2 4 / - 2`
A try catch block has
- One try block
- One or more parameterized catch blocks
- Zero or one finally block

When an exception occurs in a try block
- Remaining code in the try block is abandoned
- The first catch block that can handle the exception is executed

The try catch block terminates when
- The try block executes successfully without exception
- A catch block finishes execution
  - Which may throw its own exceptions that may be caught or not through try blocks

The finally block is called after termination of the try catch block and before the statement following the try catch block
- All paths from the associated try catch block lead to it
try {
    for (;;) {
        String nextLine = input.readLine();
        if (".".equals(nextLine)) break;
        System.out.println(Integer.parseInt(nextLine));
    }
} catch (Exception e) {
    e.printStackTrace();
}

Better style trumps over efficiency
CHECKED EXCEPTION REVIEW

- Checked exceptions
  - Uncaught exceptions must be acknowledged
Catching vs. Acknowledging

static void echoLines (int numberOfInputLines) {
    try {
        for (int inputNum = 0; inputNum < numberOfInputLines; inputNum++)
            System.out.println(input.readLine());
    } catch (IOException e) {
    
    }
}

static void echoLines (int numberOfInputLines) throws IOException {
    try {
        for (int inputNum = 0; inputNum < numberOfInputLines; inputNum++)
            System.out.println(input.readLine());
    } catch (IOException e) {

    }
}

static void echoLines (int numberOfInputLines) throws IOException {
    try {
        for (int inputNum = 0; inputNum < numberOfInputLines; inputNum++)
            System.out.println(input.readLine());
    } catch (IOException e) {
        System.out.println("Did not input "+numberOfInputLines+" input strings before input was closed.");
        System.exit(-1);
    }
}

Allowed as caller will still work, though it will have extra handling

In real-life, analogous rules exist
REAL LIFE ANALOGY

NEW DRIVER
“Give me a brake”™

Should overstate rather than understate bad side effect
Dizziness, headache

If you are bad, you should not say you are good.
People will be disappointed
If you are good, you can say you are bad
You don’t let people down

Makes sense if there is some uncertainty
Why Allow False Positives

```java
static void echoLines (int numberOfInputLines) throws IOException {
    try {
        for (int inputNum = 0; inputNum < numberOfInputLines; inputNum++)
            System.out.println(input.readLine());
    } catch (IOException e) {
        System.out.println("Did not input "+ numberOfInputLines + 
                        " input strings before input was closed. ");
        System.exit(-1);
    }
}
```

Uncertainty in this case?

No path can lead to an IO exception

In general, Java cannot tell if an exception throwing path is taken

Also method body may evolve (from stub to full implementation)
**INTERFACE ACKS BUT NOT CLASS**

```java
public void echoLines(int numberOfInputLines) throws IOException {

    try {
        for (int inputNum = 0; inputNum < numberOfInputLines; inputNum++)
            System.out.println(input.readLine());
    } catch (IOException e) {
        System.out.println("Did not input "+ numberOfInputLines + " input strings before input was closed.");
        System.exit(-1);
    }
}
```

In instance methods, must also consider variation of a method header
INSTANCE METHOD VARIATIONS

Method header

Method implementation¹

Method implementation²
INTERFACE INSTANCE METHOD VARIATIONS

- Interface
- Method header
- Acknowledge exception union (exception^1 and exception^2)
- Class^1
- Method implementation^1
  - Throws exception^1
- Class^2
- Method implementation^2
  - Throws exception^2
Real Life Analogy

NEW DRIVER
“Give me a brake”™

A car used by experienced drivers could have this sign without doing harm.
CAR DRIVEN BY EXPERIENCED AND NEW DRIVER

- Car
  - Student Driver
    - New Driver Warning
  - Experienced Driver
    - New Driver Warning
MEDICINE USED BY REACTIVE AND NON REACTIVE PERSON

- Person¹
  - Gets Dizzy
- Person²
  - Gets Dizzy
- Medicine
  - Gets Dizzy
Instance Method Variations Throwing Different Exceptions

Method Header

Method implementation¹
- Throws exception¹

Method implementation²
- Throws exception²

Acknowledge exception union (exception¹ and exception²)
**Exceptions**

- Support typed errors
- Provide a way to customize error handling
  - By default Java will terminate program
- Allows more efficient error processing
- Allows separation of error-handling and normal-processing code
- Allows error handling and error detection to be in separate classes and methods
  - Without passing legal non erroneous values (null/-1)
- Allows separate classes to provide error UI
Exceptions

- Errors may be internal or external
- Exceptions allow custom error handling to be done while following software engineering principles
- Try and catch blocks allow programmers to easily separate error handling and non error handling code
- Sometimes error handing should be distributed among error detecting method and its callers
- Need a way for error information to be passed to caller, that is, propagate error information
- Checked and unchecked exceptions