Exception Handling Basic
-- from generic idea to Java

Daily Idea

In our life, there are something that we don't expect to happen but they happens; or something that's rare but still happens. We can consider these things exception.

- Blackout
- Snow day

We react to these things (or called event) in some way to prevent loss, this can be called exception handling.

Daily Idea

- Example: Snow day

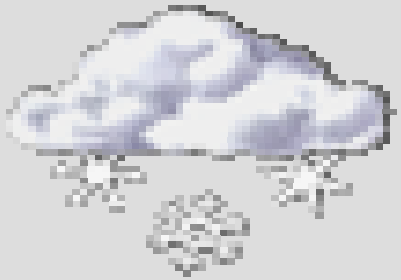
Consider a school-student system. When there is a snow storm coming up. Different parties (school & student) should act properly to prevent loss.

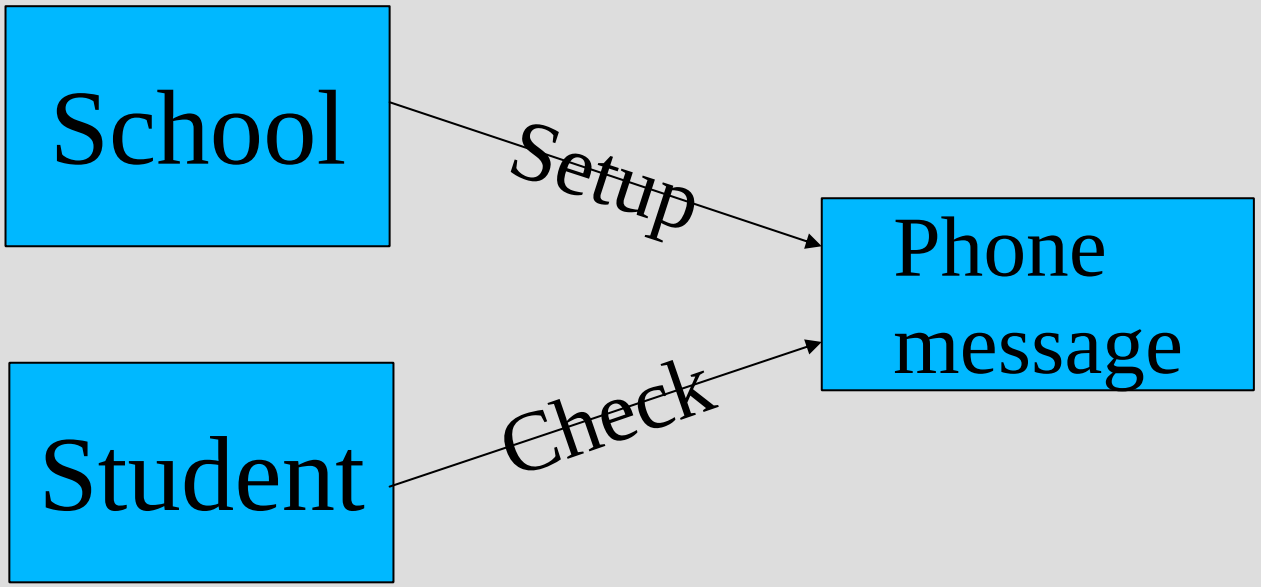
- School: decide if they should close for the day and setup a voice message in its phone to notify the temporary of school.
- Student: check by phone if school close for the day.

Daily Idea

- Example: Snow day

The reaction of both parties to communicate, such that the whole system react correctly to prevent loss:





Daily Idea

- Example: Snow day

The party that do not communicate correctly or react correctly can gain loss for them-self or the other parties.

Schools that does not close for extremely strong storm:

- may gain a law sue from students or employees

Students that come to school without checking:

- may waste time
- higher probability of car accident

Exceptions in Software

Different parties of the software should communicate together and react together correctly to avoid loss as much as possible.

For the term “different parties” we mean: modules or routines/methods/functions that have different responsibilities and work together.

Exceptions in Software

In software systems, there are many situation that can be considered exceptions:

- division by zero
- number underflow or overflow
- not enough memory
- hardware failure
- try to access a file but file missing
- ...

Exceptions in Software

- try-catch mechanism

try-catch mechanism is used by currently many programming languages to communicate each other and handle exceptions:

- Java (we will use this to study for now)
- C++
- Object Pascal
- Matlab
- <should be more but I don't know>

Java Exception Handling

- Example: division by zero

```
public class Main {

    private static float div(float a, float b){
        // a wrapper to computer a/b
        return a/b;
    }
    public static void main(String[] args) {
        // TODO code application logic here
        float result1=div(3,4);
        float result2=div(3,0);
        float result3=div(result1+result2,result2);
        System.out.println("result1="+result1+","+
                           " result2="+result2+
                           ", result3="+result3);
    }
}
```

Java Exception Handling

- Example: division by zero

```
run:
result1=0.75, result2=Infinity, result3=NaN
BUILD SUCCESSFUL (total time: 0 seconds)
```

The program compiles and runs, but we may not be happy with the results:
If the div() routine is directly or indirectly used by more other methods to get more results, we may want an explicit notification to calling method instead of seeing more Infinity or NaN.

Java Exception Handling

- Example: division by zero

```
private static float div(float a, float b) throws Exception{
    // a wrapper to computer a/b
    if (b==0) throw new java.lang.Exception("division by zero");
    return a/b;
}
public static void main(String[] args) {
    // TODO code application logic here
    try{
        float result1=div(3,4);
        float result2=div(3,0);
        float result3=div(result1+result2,result2);
        System.out.println("result1="+result1+","+
                " resutl2="+result2+
                ", result3="+result3);
    }catch(Exception e){
        System.out.println(e.toString());
    }
}
```

Java Exception Handling

- Example: division by zero

```
run:
java.lang.Exception: division by zero
BUILD SUCCESSFUL (total time: 0 seconds)
```

Now we will not see Infinity and NaN, but notified explicitly of exception.

Java Exception Handling

- Example: division by zero

Question:

Why not just let div() method handle the exception by itself, but throw an Exception object to its caller and let its caller handle it?

Java Exception Handling

- Example: division by zero

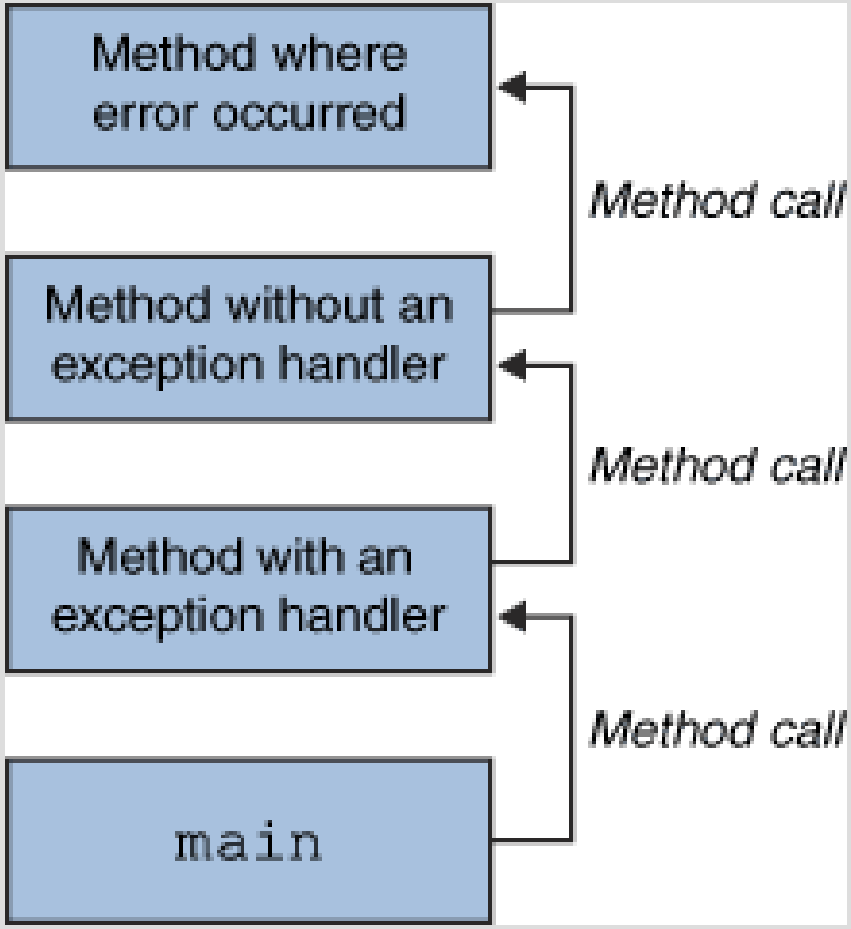
Question:

Why not just let div() method handle the exception by itself, but throw an Exception object to its caller and let its caller handle it?

Answer:

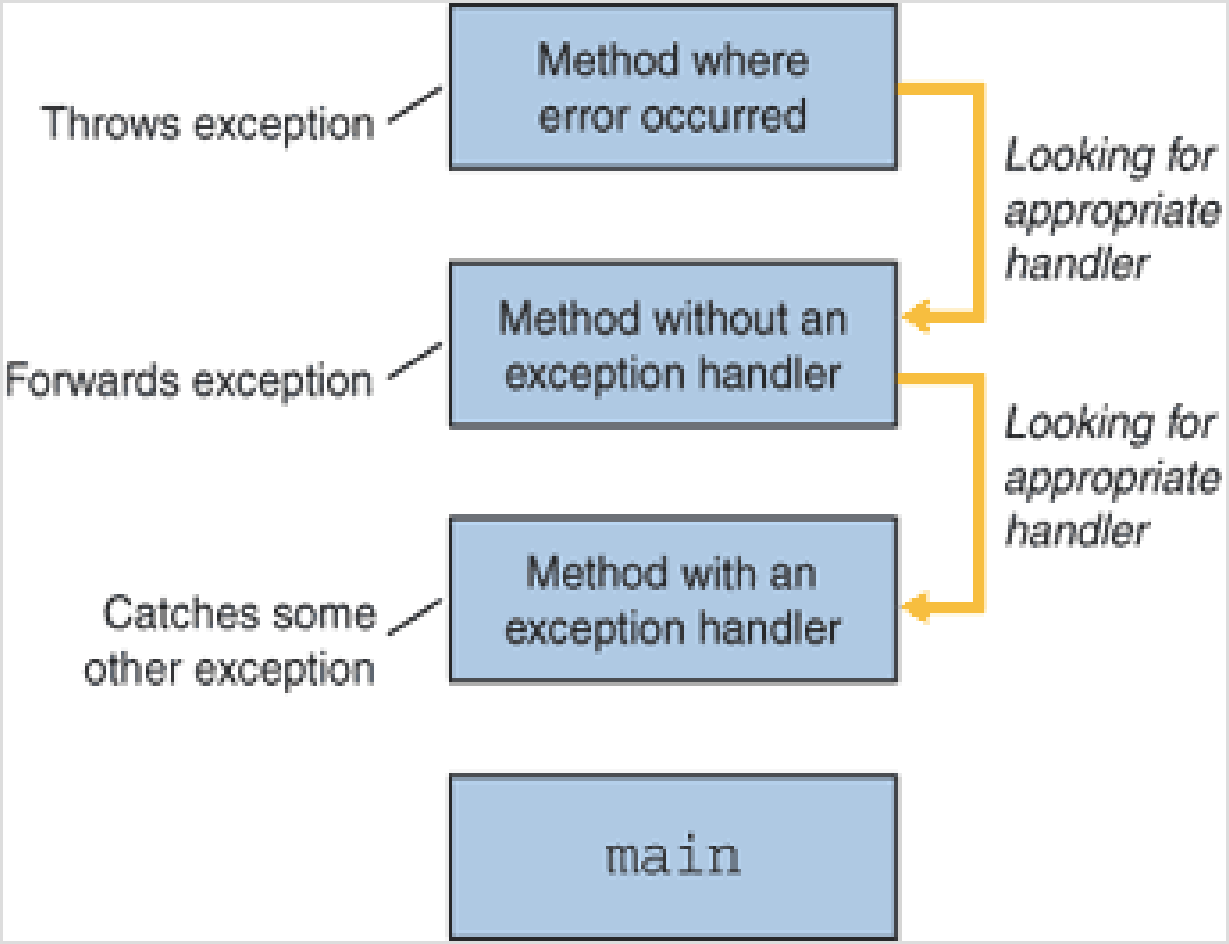
Because the implementer of div() may not know what the user of div() want in case of exception. It shall leave it for the caller.

Call Stack view of throwing



Sun's lesson of Exception
http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html

Call Stack view of throwing



Sun's lesson of Exception
http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html

Call Stack view of throwing

A handler can either:

- Really handle the exception, or
- throw another Exception

Java Exception Handling

- finally block

Some times there will be code that should be executed doesn't matter if exception is caught or not. For example, resources need to be released even if some operations to the resource fails.

These code can be put into a finally block.

Java Exception Handling

- finally block

```
try{ // the code that try to be run
}
catch(...){// first handler
}
....
catch(...){// last handler
}
finally{// the code that will be run anyways
}
```

Java Exception Handling

- finally block example

```
public void writeList() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("OutFile.txt"));
            for (int i = 0; i < SIZE; i++) out.println(vector.elementAt(i));
    } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println(e.getMessage());
    } catch (IOException e) {
            System.err.println(e.getMessage());
    } finally {
        if (out != null) out.close();
     }
}
```

Java Exception Handling

- finally block example

```
public void writeList() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("OutFile.txt"));
            for (int i = 0; i < SIZE; i++) out.println(vector.elementAt(i));
    } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println(e.getMessage());
    } catch (Exception e) {
            System.err.println(e.getMessage());
    } finally {
        if (out != null) out.close();
     }
}
```

Note this section of code may not compile because it is just for demo purpose and lack of pre-requested class definitions. The code is the result of simplifying code from
http://java.sun.com/docs/books/tutorial/essential/exceptions/putItTogether.html

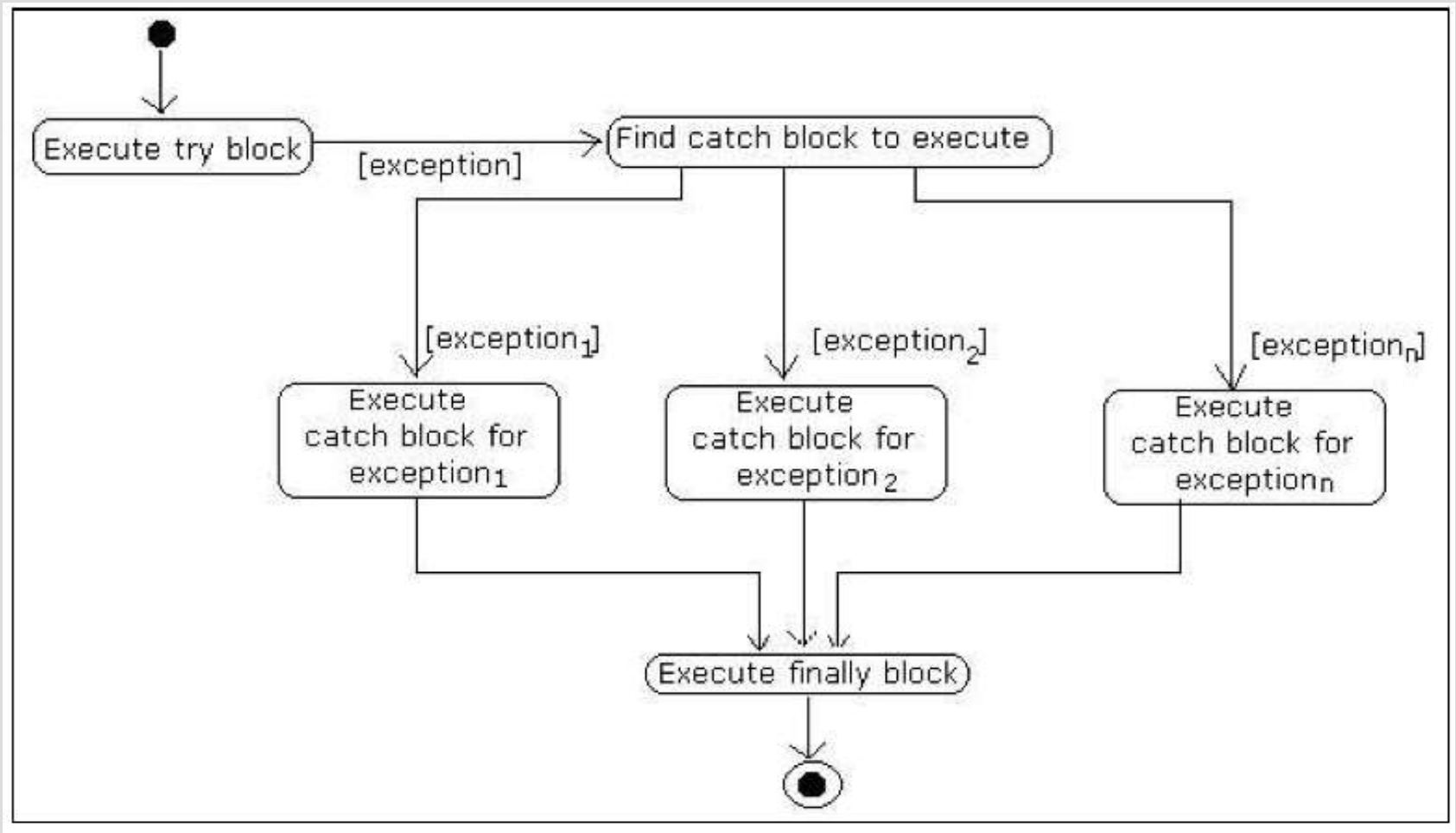
Java Exception Handling

- finally block example

```
public void writeList() {
    PrintWriter out = null;
    try {
            out = new PrintWriter(new FileWriter("OutFile.txt"));
            for (int i = 0; i < SIZE; i++) out.println(vector.elementAt(i));
    } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println(e.getMessage());
    } catch (Exception e) {
            System.err.println(e.getMessage());
    } finally {
        if (out != null) out.close();
     }
}
```

Note this section of code may not compile because it is just for demo purpose and lack of pre-requested class definitions. The code is the result of simplifying code from
http://java.sun.com/docs/books/tutorial/essential/exceptions/putItTogether.html

Exceptions are classes in Java



from
http://www.javapassion.com/javaintro1/slides/JEDI%20Slides-Intro1-Chapter12-Basic%20Exception%20Handling.pdf

References

Sun's lesson of Exception
http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html

Java Intro Programming (With Passion!) Online Boot Camp
http://www.javapassion.com/javaintro1/