COSC 2011 3.0 Fundamentals of Data Structures
Section N, Winter 2001

Class Notes: **Exceptions**

By: Bill Kapralos

In a Perfect World:

- User input contains no errors.
- The files a user chooses to open always exist.
- Code never has bugs and compiles first time.

In Reality, this is not so!:

- Errors are very unpleasant.
- If a user loses work he or she did while using a program, they may turn away from your program forever! At the very least you should:
  1. Notify the user.
  2. Save all work.
  3. Allow the user to exit the program gracefully.

Traditional Methods of Dealing with Errors Include:

- Output an Error Message and exit the program.
  - The "Lazy Way Out"!
  - Avoid this when possible.
- Return a special error code that the calling method analyzes. For example, return -1 to indicate the end-of-file condition.
  - However, may not always be able to return a valid error code as there may be no way of distinguishing valid and invalid error codes. For example, the value -1 may be a perfectly valid return type.
  - Increases and complicates the code required to handle the many possible error conditions.
- Assertions

- Will inform the user of any erroneous input but usually, nothing will be done to correct the problem.

In the event of an error occurring in a Java program, Java does not rely on traditional methods to deal with the problem. Java relies on a form of **Error Trapping** called **Exceptions**.

Exceptions:

- Unexpected events that occur during the execution of a program.
- Can be the result of an error condition or simply, unanticipated user input.
- Cover a wide spectrum of program events, from running out memory, to dropping a network connection.
- Originate when a piece of Java code in a Java method encounters a problem during program execution and **throws** an *exception object*, which identified with a descriptive name.
  - When the problem is encountered, an exception object is created.
  - The exception object is given to the run-time system.
  - The object contains information about the exception, including the type of exception and the state of the program when the error occurred.

Two types of exceptional conditions, **Errors** and **Exceptions**:

Error Represents a condition that the program should not try to recover from. Describe internal errors and resource exhaustion inside the Java Run-Time system. Little you can do if such an internal error occurs except for notify the user and attempt to terminate the program gracefully.

Exception Code errors, Properly handled exceptions allow the program to continue.

Exceptions are thrown:

- By code that encounters the unexpected condition.
- By the Java Run-Time Environment should it encounter an unexpected condition, such as running out of memory, or invalid array indexing.
- When a method is declared, you must specify any exceptions it may throw!
  - A method may throw multiple exceptions.

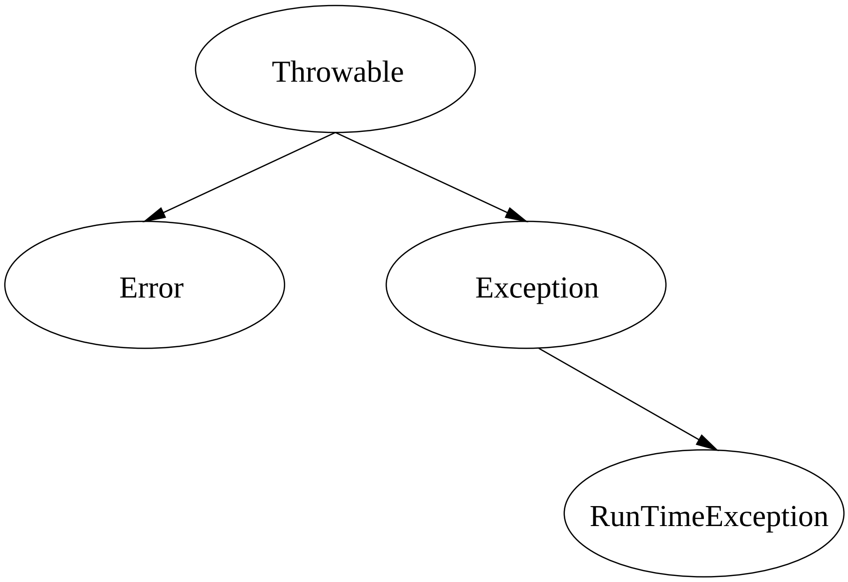

Figure 1: Class Hierarchy of Throwable Objects. Exceptions and Errors are sub-classes of the Throwable class.

- Lets the compiler know which exceptions to prepare for.
- Lets the user of the method know what to expect in terms of error conditions when using the method.

After the exception has been created and thrown:

- Java Run Time system will find some code to *handle* the exception (this code is known as the *handler*).
  - The set of possible handlers for an exception are the set of methods in the **Call Stack**. e.g. All methods in the Call Stack "below" the method where the exception took place.
  - Searches backward through the Call Stack until it finds the appropriate handler (see figure 2).
  - If no appropriate handler is found, the program terminates.

Throwing an Exception:

1. Find an appropriate Exception class or declare your own which inherits from the Exception class or any of its sub-classes.

2. Make an object the above class.

3. Throw it with the keyword **throw**.

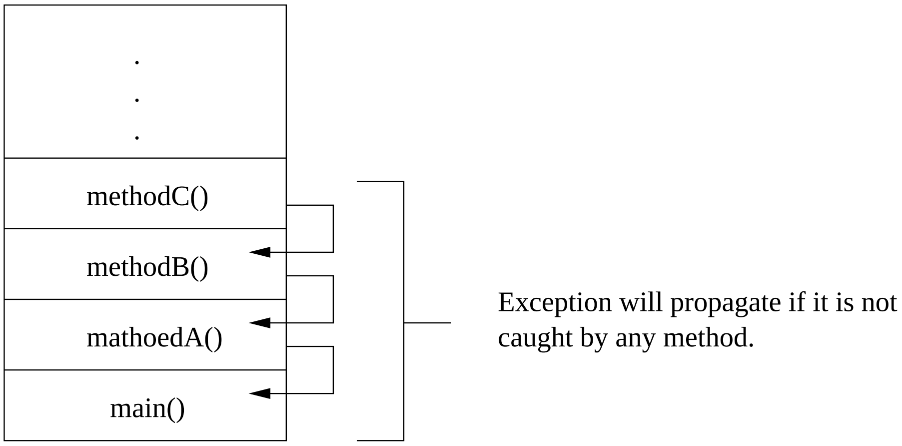

Figure 2: Java Call Stack. Every method which is called is allocated a portion of memory in the **Java Call Stack**. The method uses this memory to declare variables etc. Method calls go from bottom to top - e.g. in this example, main() is called first, followed by methodA(), methodB() and methodC(). The top-most method, is the currently running method. If an exception is thrown by the running method, it will propagate down the Call Stack until a *handler* is found to take care of the exception, or the program will eventually exit.

When using a method which may throw an exception:

- Catch the exception.
- Ignore the exception (don't catch it). However, you must then specify that your method throws the exceptions you choose to ignore.

General methodology for Catching Exceptions:

- **try** to execute some code fragment that might throw an exception.
- If an exception is thrown, then it is caught, by having the flow of control transfer to a pre-defined **catch** block. The exception is dealt with in the catch block.

You provide for handling exceptions by enclosing the statements that may throw an exception in a block of code that is tagged with the keyword **try** and followed by one or more blocks tagged with the keyword **catch** statements and an optional **finally** statement.

- Multiple catch statements are permitted to allow for a variety of exceptions to be caught.

- The order of catch clauses must always go from the most specific exception class to the most general. For example, consider "catching" two exceptions: FileNotFoundException and IOException. Since FileNotFoundException is a more specific form of IOException, the catch block catching FileNotFoundException must come before the catch block catching IOException.
- finally Block:
  - Optional. May or may not be included.
  - Always executed regardless of whether there is an exception or not.
  - Usually included to make sure some "clean up" is performed (e.g closing opened files, returning any system resources etc.).
  - If a finally clause includes a transfer of control (e.g. return, break, continue or throw) then that statement overrides any transfer of control initiated in the try or catch clauses.

```
try
  statements which may throw exceptions here.

catch(...)
  Deal with exception here.

catch(...)
   Deal with exception here.

...
finally
   Code to be executed in the finally block goes here.
```

Exceptions on Top of Exceptions: In a "real world" programming situation, the code in a try-catch block may itself cause an exception. If this happens, the new exception replaces the original one and the JVM tries to find a handler for the new exception.