Where are We?

Java: read chapters 13, 15, and 17
This should be mostly a review…with the exception perhaps of exceptions.

Assignment #8: see website

Translator

Problem: Build a simple translator from arithmetic expressions to a simple stack machine.

The translator accepts the following language:

```
G:  <expression> ::= <mulexp> '+' <mulexp>
                  |  <mulexp> '-' <mulexp>
    <mulexp>     ::= <rootexp> '*' <rootexp>
                  |  <rootexp> '/' <rootexp>
    <rootexp>    ::= '(' <rootexp> ')'
                  |  number
```

The translator generates the following stack machine language:

```
G':  <comlist>    ::= <comlist> <command> | <empty>
     <command>    ::= <arithmetic> | <stack>
     <arithmetic> ::= add | subtract | multiply | divide
     <stack>      ::= push number | pop
```

Base your implementation on the calculator code given in the book.

Translator

- Recursive descent parser…one function for each non-terminal
- Given the expression (1+2)*3 your translator should produce:

```
push 1.0
push 2.0
add
push 3.0
multiply
```

- Note: it is assumed that the arithmetic commands pop the values off the stack that they use.
- Java code for calculator is available on the web (don't copy it from the book)

Java Exceptions

The term *exception* is shorthand for the phrase "exceptional event."

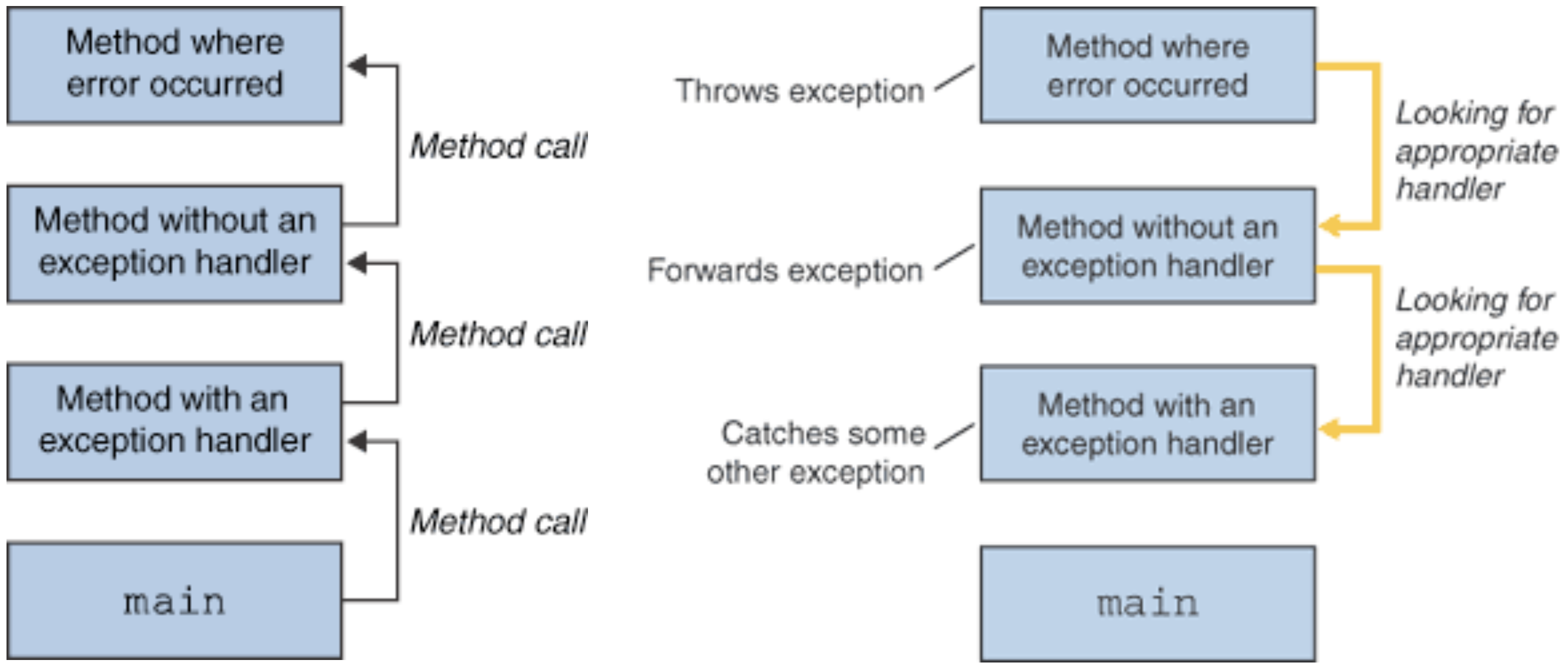
Def**:** An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. Handlers *catch exceptions.*

Source: http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html

Java Exceptions


Method where error occurred
Method without an exception handler
Method with an exception handler
main
Method call
Method call
Method call
Throws exception
Forwards exception
Catches some other exception
Method where error occurred
Method without an exception handler
Method with an exception handler
main
Looking for appropriate handler
Looking for appropriate handler

Java Exceptions

```
 try {
   // Perform work here
 } catch (Exception e) {
   // Log the exception and continue
   System.out.println("Unexpected exception", e);
}
```

```
// In some function h()
throw new Exception("optional text here");
```

Checked vs. Unchecked Exceptions

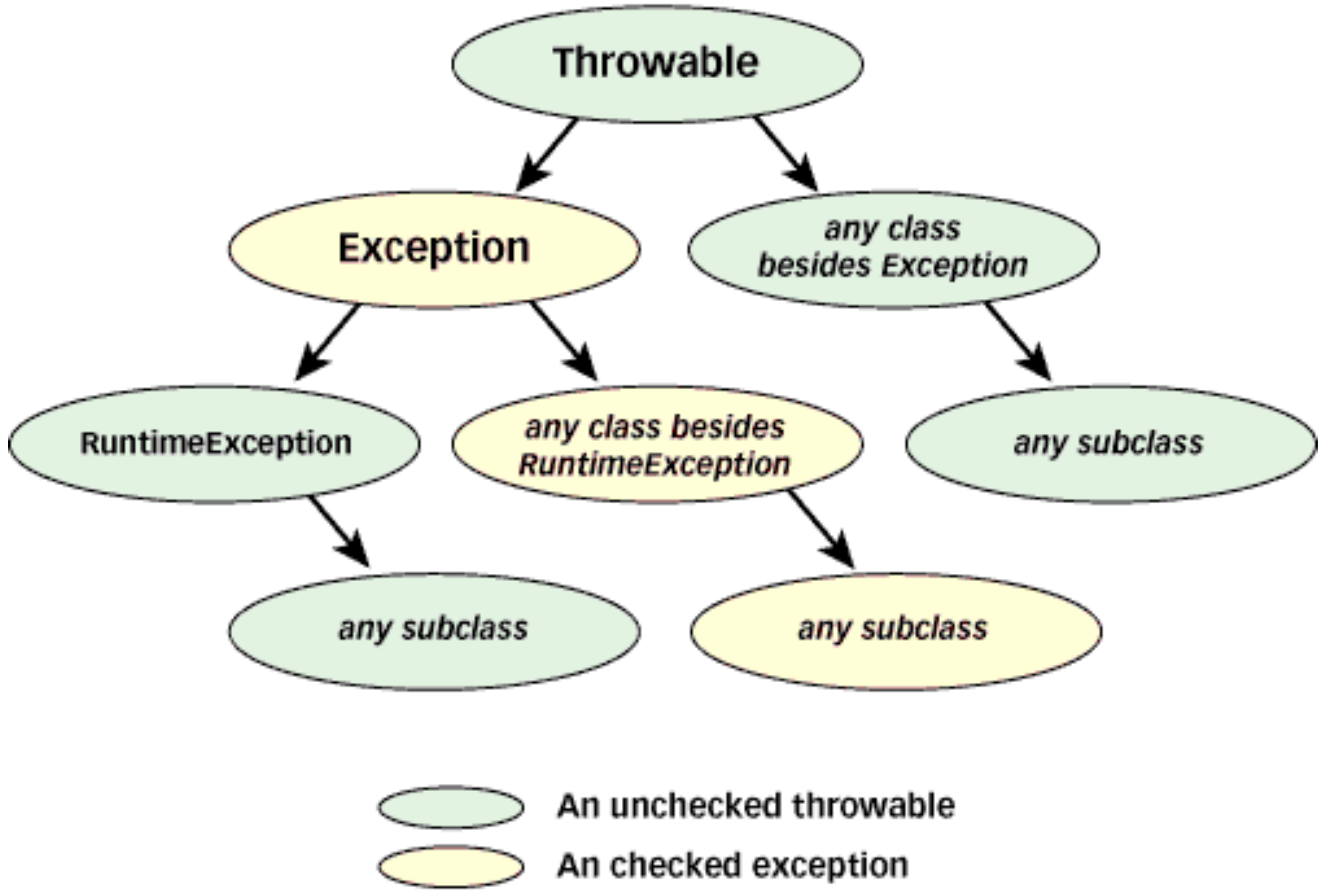
- Unchecked exceptions :
  - represent defects in the program (bugs) - often invalid arguments passed to a non-private method. To quote from The Java Programming Language, by Gosling, Arnold, and Holmes :

    "Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time."
  - a method is not obliged to establish a policy for the unchecked exceptions thrown by its implementation (and they almost always do not do so)

Checked vs. Unchecked Exceptions

- Checked exceptions :
  - represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files)
  - a method is obliged to establish a policy for all checked exceptions thrown by its implementation (either pass the checked exception further up the stack, or handle it somehow)

Java Exceptions



Java Exceptions

The Exception class behaves just like any other class:

```
Class MyException extends Exception {
        MyException (List items) {…};
        void printList() {…}
}
```