Another Recursive Descent Example

- **Goal**: Determine if the brackets ( ) [ ] { } on a line are balanced and properly nested.
- **Examples**:
  - legal: ( ) [ ( ) ]
  - legal: ( ) [ ]
  - illegal: ( )
  - illegal: ( )

Recursive definition for LegalExp
- The empty string is a LegalExp
- If E is a LegalExp then so are (E), [E], and {E}
- One or more LegalExps on a line make a LegalExp

Step 1: Build a Tokenizer

- We want to divide an input line (a String) into tokens.
- The Java API on the Web includes a java.util package.
- This package contains java.util.StringTokenizer.
- StringTokenizer has a nextToken() method that throws NoSuchElementException if it runs out of tokens.
- StringTokenizer has a nextToken() method that throws NoSuchElementException if it runs out of tokens.
- If E is a LegalExp then so are (E), [E], and {E}
- One or more LegalExps on a line make a LegalExp

Using StringTokenizer

- Can StringTokenizer be used directly? Almost...
  - We want to skip all "uninteresting" tokens.
  - Would like an eol token.

- We can alter StringTokenizer to do these extra things by using either inheritance or aggregation.

Tokenizer Code

- Aggregation:
  - We make our own tokenizer by using a StringTokenizer within our tokenizer class.
  - Our class has only the methods that we choose to write.
  - nextToken()
  - pushBack()

```java
class MyTokenizer {
    private StringTokenizer tokenizer;
    private String currentToken;
    private boolean pushed;
    public static String PARENS = "()[]{}";
    public MyTokenizer (String inputString) {
        tokenizer = new StringTokenizer(inputString,PARENS,true);
        currentToken = null;
        pushed = false;
    }
    public void pushBack () {
        pushed = true;
    }
    public String nextToken () {
        if (pushed) {
            pushed = false;
            return currentToken;
        }
        try {
            do {
                currentToken = tokenizer.nextToken();
            } while (PARENS.indexOf(currentToken) == -1);
        } catch (NoSuchElementException e) {
            currentToken = "eol";
        }
        return currentToken;
    }
}
```
Step 2: Build a Parser

- Follow the recursive definition
- The empty string is a LegalExp
- If E is a LegalExp, then so are (E), [E], and {E}
- One or more LegalExps on a line make a LegalExp

```java
class LegalExpParser {
    MyTokenizer in;
    public void legalExp () {
        String matcher;
        String token = in.nextToken();
        while ("()\{\[".indexOf(token) != -1) {
            if (token.equals("(")) matcher = ");
            else if (token.equals("[")) matcher = "]
            else matcher = "}
            legalExp();
            token = in.nextToken();
            if (!token.equals(matcher)) error;
            token = in.nextToken();
        }
        in.pushBack();
    }
}
```

The Rest of the Code

```java
public void eval (String inputString) {
    in = new MyTokenizer(inputString);
    legalExp();
    if (in.nextToken().equals("eol")) return;
    error;
}
```

How Recursion Works

```java
int fact (int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Stack Frames

- One slot for each parameter (values computed and pushed by caller)
- One slot for return address (where to continue execution after method is done)
- One slot for each local variable (allocated by callee)
- Temporaries are pushed/popped on top of frame

Typical Stack Frame

- How does a method-call know where to return to?
  - Save a return address before making the call
- How does the callee return the result to the caller?
  - Result is left on top of the stack
- How do we keep track of all this information?
  - Organize the information for each call into a frame

FBR (Frame Base Register) points into current (topmost) frame

Values within frames are addressed via offsets from FBR

SP (Stack Pointer) moves up and down as temporary values are pushed and popped

How Exceptions Work

```java
int foo (int a, int b, int c) {
    int w,x,y,z;
    ...
}
```

- What a Java program does when an exception occurs
- When running a Java application
  - There is a “first” frame that catches all exceptions
  - It prints some semi-helpful information and then halts