Lists, Stacks, and Queues (plus Priority Queues)

The structures *lists*, *stacks*, and *queues* are composed of similar elements with different operations. Likewise, with mathematics: $(\mathbb{Z}, +, 0)$ vs. $(\mathbb{Z}, *, 1)$

List	Stack (LIFO)	Queue (FIFO)	Priority Queue
create empty str.	√	√	√
test emptiness	√	√	√
add elem to head	√ [push]	add elem to end	add elem
access head elem	√ [top]	access front elem	access highest prio. elem
access tail	[pop] top of stack	remove front elem	remove highest prio. elem
modify head			
modify tail			

Implementation a Stack via Wrapping a List

```
class Stack  {
   private List c;
   Stack(final List x) {this.c = x;}}
   Stack() {this.c=null;}
   public boolean isempty () {return this.c == null;}
   public void push(int x) {this.c = new List(x,this.c);}
   public int top() {return this.c.hd;}
   public void pop() {this.c = this.c.tl;}
   }
```

There are two bugs in this code!

Functional Stack

```
class FStack {
  private final List c;
  FStack() { this.c = null; }
  FStack(final List l) { this.c = l; }
  static boolean isempty(final FStack l) { return l.c == null; }
  static FStack push(final int a, final FStack l) {
    return new FStack(new List(a, l.c)); }
  static FStack pop(final FStack l) { return new FStack(l.c.tl);
  static int top(final FStack l) { return l.c.hd; }
}
```

Note “final FStack l” means that “l” is never modified
but its contents may be.

Exceptional Circumstances

Two exceptional circumstances in FStack executions:

- pop of an empty stack
- top of an empty stack

How to Deal With them:

1. figure out and deal with usual situations.
2.
   - (Old Style Programming) let weird stuff happen otherwise:
   - (Newer Style Programming) deal with unusual situations explicitly.

Exceptions - Cont.

Safety conscious (Using Exceptions):

1. figure out unsafe situations.
2. deal with those first.
3. once safe do something usual.

```
static int top(final FStack l) throws Exception {
if (l == null) throw new Exception();
return l.c.hd;}
```

Catching Exceptions.

Specifying what to happen when the function raises an exception:
The construct “try p catch (Exception e) q;”

```
   try {
          int x =  top(new FStack());
       } catch Exception e) {
     system.out.println(''oh oh!'');
                               }
```

The exceptions will propagate until caught.
Note. in $\{p_1 p_2\}$, if p_1 throws exception, p_2 is not executed. (like return)

Exception Handling in Caml and C.

- **Caml**: "throw e" is written "raise e".
  try $\cdots$
  with $\_ \rightarrow \cdots$
- **C**: There are no exceptions in C.
  some constructs like "long jumps" have some similarities.

Exception - Cont.

- We can have new classes of exceptions (you can make your own)
- We can catch multiple exceptions

- **try-catch-finally**:
  The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs.

- Error Messages in Java:

  ```
  System.out.println(1/0);
  ```

  throws the exception:

  ```
  java.lang.ArithmeticException: / by zero
  ```

Objects and Classes - Dynamic Methods

A *class* is a type plus some functions and constructors on that type.

T: is a class; f : a static method in T; $a : T$
Consider a call of f: $T.f(b_1, \cdots, a, \cdots, b_n)$
We can distinguish between a and other arguments: $a.f(b_1, ..., b_n)$
This type of method is called dynamic

```
void push (final int a) {
c = new List(a, c);}
void pop () {
c = c.tl;}
```

We call these methods with p.pop(); p.push(5);

A dynamic method belongs to an object.
A static method, in contrast, belongs to a class.

Dynamic Methods - Cont.

```
Stack p = new Stack();
p.push(5);
p.push(6);
System.out.println(p.top());
p.pop();
```

The printed result of the above code is 6.
If “System.out.println(p.top());” is used after “p.pop();” then the result is 5.

Empty stack is an object with $c == null$.
A common error is to write a dynamic method:

```
List f () {
 if (this == null) ...}
```

“(this == null)” always false!: the method f cannot be called when the object is null.

Static Fields

When we modify a static field, it is modified for all the class (global fields!).

```
class M {
 static int mem;
...
```