Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-four)

# 4.9: Classic Data Structures Using Objects: A Stack

A stack is a data structure that operates like a bunch of cars parked in a driveway for a party: the last car to arrive must be the first car to leave (because it's blocking all the others!). A data structure like this is often called a LIFO (Last-In, First-Out) structure. You can only add or remove objects from a stack from one end, the "top." Adding objects is called *pushing* them onto the stack, while removing them is called *popping* them off of the stack.

Stacks are handy in many interesting situations. For example, in the following algorithm for reversing the characters in a stream of input:

```
while (there is a character to read)
    push the character onto a stack;

while (there are characters on the stack)
    pop off a character and print it;
```

The concept of a stack is a perfect candidate for becoming a class. It has a well-defined interface (push( ) and pop( )) and some rules to enforce (you can only take data from the top, you can only remove as many elements as you insert). Here is a simple implementation of a stack that holds characters:

```
import java.io.*;

public class CharStack
{
  private char[] m_data;           // See Note #1 below

  private int m_ptr;

  public CharStack(int size)
  {
      m_ptr = 0;                   // Note #2
      m_data = new char[(size > 1 ? size : 10)];
  }

  public void push(char c)
  {
      if (m_ptr >= m_data.length) // Note #3
      {
         // Grow the array automatically
         char[] tmp =
            new char[m_data.length * 2];

         System.arraycopy(m_data, 0,
                          tmp, 0,
                          m_data.length);
         m_data = tmp;
      }
```

```
        m_data[m_ptr++] = c;
    }

    public char pop()              // Note #4
    {
        return m_data[--m_ptr];
    }
    public boolean hasMoreElements()
    {
        return (m_ptr != 0);
    }

    // Note #5
    public static void main(String[] argv)
        throws IOException
    {
        CharStack s = new CharStack(10);
        int i;
        while ( (i = System.in.read()) != -1 )
        {
           s.push((char) i);
        }
        while (s.hasMoreElements())
        {
           System.out.write(s.pop());
        }
        System.out.println();
      }
}
```

---

**Note #1:** Note the use of the "decorated" names m_ptr and m_data to denote member variables. This makes their memberness obvious by inspection, and can make code easier to understand. If you use names like this, be sure to use them consistently!

**Note #2:** The constructor won't let you try to create a stack with an initial capacity of less than one element. This is an example of the kind of fail-safety you should try to build into all of your classes: make it impossible for them to become internally corrupted.

**Note #3:** When we've filled up our initial storage array, these few lines of code allocate a new one and copy the contents of our old one. Again, code that uses CharStack won't risk CharStack's running out of storage space; CharStack takes care of everything.

**Note #4:** pop( ) can fail if m_ptr is zero. We haven't talked about the right way to handle this yet (using Exceptions), so I'm ignoring it. In a real class, you'd never ignore this possibility!

**Note #5:** This main( ) function creates a CharStack and lets you type in some characters. When you send an end-of-file character (Ctrl-Z on a line by itself for Windows; Ctrl-D for Unix), it prints the characters reversed. I've placed this function inside CharStack for convenience, but it could just as easily be inside another class. It's actually a good habit to get into to write main( ) functions in many of your

classes that do nothing but test the class they appear in. Then you'll always have a test driver handy when you need to check some of your code, and this test function can serve as a usage example too.

The Java API provides a Stack class (java.util.Stack) into which you can insert objects but not chars, ints, etc. It also provides a Vector class (growable array) and a Hashtable class (associative array).