A simple stack with limited size

In preparation for writing a second iterator, we write a class that implements a stack of limited size.

```
import java.util.EmptyStackException;
import java.util.Iterator;
import java.util.NoSuchElementException;

/** An instance is a stack */
public class Stack<E> implements Iterable<E> {
   private E[] b;  // stack values are b[0..h-1],
   private int h;  // with b[h-1] at the top

   /** Constructor: a stack of at most m values */
   public Stack(int m) {
      b= (E[]) new Object[m];
   }

   /** Push e onto the stack. if there is no room,
     * Throw a RuntimeException("no space") */
   public void push(E e) {
       if (h == b.length)
          throw new RuntimeException("no space");
       b[h]= e;
       h= h+1;
   }

   /** Pop and return the top stack value. Throw an
     * EmptyStackException if the stack is empty. */
   public E pop() {
      if (h == 0) throw new EmptyStackException();
      h= h - 1;
      return b[h];
   }
   /** = the size of the stack */
   public boolean size() {
      return h;
   }
}
```

The class invariant indicates that the stack has h values, which are in b[0..h-1], with b[h-1] the top stack.

The parameter of the constructor is the maximum size m of the stack. Look at the way the array is created. An Object array is created and then cast to E[]. That's how you have to do it.

There's the usual push operation on a stack; note the exception that is thrown if an attempt is made to push element number h+1 on the stack. And there is the usual pop operation, with an exception if the stack is empty. Finally, we write a method method that gives the size of the stack.

This is a barebones implementation of a stack. It has just the minimal stuff so that we can write an iterator as an inner class. That's the next video.