```
public class EliminationBackoffStack<T> extends LockFreeStack<T> {
  static final int capacity = ...;
  EliminationArray<T> eliminationArray = new EliminationArray<T>(capacity);
  static ThreadLocal<RangePolicy> policy = new ThreadLocal<RangePolicy>() {
    protected synchronized RangePolicy initialValue() {
      return new RangePolicy();
    }

  public void push(T value) {
    RangePolicy rangePolicy = policy.get();
    Node node = new Node(value);
    while (true) {
      if (tryPush(node)) {
        return;
      } else try {
        T otherValue = eliminationArray.visit(value, rangePolicy.getRange());
        if (otherValue == null) {
          rangePolicy.recordEliminationSuccess();
          return; // exchanged with pop
        }
      } catch (TimeoutException ex) {
        rangePolicy.recordEliminationTimeout();
      }
    }
  }
}
```

FIGURE 11.8 The EliminationBackoffStack<T> class: This push() method overrides the LockFreeStack push() method. Instead of using a simple Backoff class, it uses an EliminationArray and a dynamic RangePolicy to select the subrange of the array within which to eliminate.