理论上java因为有垃圾回收机制（GC）不会存在内存泄漏的问题(这也是java被广泛使用于服务器端编程的一个重要原因)，然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被GC回收，因此也会导致内存泄漏的发生。例如Hibernate的Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象，如果不及时关闭或者清空一级缓存就可能导致内存泄漏。下面的例子中的代码也会导致内存泄漏。

```
import java.util.Arrays;
import java.util.EmptyStackException;

public class MyStack<T> {
    private T[] elements;
    private int size = 0;

    private static final int INIT_CAPACITY = 16;

    public MyStack() {
        elements = (T[]) new Object[INIT_CAPACITY];
    }

    public void push(T elem) {
        ensureCapacity();
        elements[size++] = elem;
    }

    public T pop() {
        if(size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }

    private void ensureCapacity() {
        if(elements.length == size) {
            elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }
}
```

上述代码看似没有问题，但是其中的pop方法却存在内存泄漏的问题，当我们用pop方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用。在支持垃圾回收的语言中，内存泄漏是很隐秘的，这种内存泄漏其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理这个对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下引发Disk paging（物理内存与硬盘的虚拟内存交换数据），甚至造成outofmemoryError。